R. Lullus
C. Lorenz

Programmieren in Maschinensprache 6502

EDITOR

ASSEMBLER

BINDER

DISASSEMBLER

Genaue Befehlsbeschreibung mit Beispielen

ISBN 3-921682-45–2

1. Auflage 1980

Gedruckt in der Bundesrepublik Deutschland – Printed in West-Germany – Imprime' en RFA.

R. Lullus
C. Lorenz

Programmieren in Maschinensprache 6502

EDITOR

ASSEMBLER

BINDER

DISASSEMBLER

Genaue Befehlsbeschreibung
mit Beispielen

Vorwort

Die Vorteile der Programmierung in Maschinensprache liegen klar auf der Hand:

1. Effiziente Speicherausnutzung
2. Hohe Geschwindigkeit
3. Besonders Instruktiv

Das Ihnen hier vorliegende Buch soll dem 6502 Programmierer eine Hilfe bei der täglichen Arbeit sein. Nach einer kurzen Beschreibung der Architektur des 6502 Prozessors folgt eine ausführliche Behandlung jedes 6502 Maschinenbefehls mit genauer Beschreibung der Funktion und ein Beispiel. Das Studium dieser Befehle bildet die Grundlage für Ihre zukünftige Programmierung in Maschinensprache, die Ihnen ungeahnte Möglichkeiten eröffnen wird.

Am Schluß des Buches finden Sie dann noch ein komplettes Programmentwicklungssystem bestehend aus Editor, 2 Pass-Assembler, Binder und Disassembler. Die Programmlistings für Commodore PET 2001 8K und für die CBM Versionen sind beigefügt.

Ich wünsche Ihnen, lieber Leser bei der Verwendung dieses Buches viel Erfolg.

Frühjahr 1980 Die Verfasser

Inhaltsverzeichnis

1. Programmieren in Maschinensprache

Architektur des 6502 CPU

Der 6502 Microprozessor hat eine bestimmte Anzahl von Registern, welche dazu benutzt werden um die Daten zu speichern.

Es handelt sich dabei um die folgende Register:

1. Programmzähler (Programmcounter)
   Dieses 16 Bit Register enthält immer die Adresse des Befehls, der als nächstes ausgeführt wird.

2. Das Status Register
   Dieses 8 Bit Register enthält die verschiedenen "Flags" (Zeichen) die zur Steuerung des CPU verwendet werden. Die einzelnen Bits dieses Registers haben folgende Bedeutung.

Bit	Beschreibung
0	Carry-Bit Dieses Bit wird gesetzt, wenn die vorhergehende Addition einen Übertrag (Carry) erzeugte. D.h., daß eine Zahl größer 255 entstanden ist.
1	Zero-Bit Dieses Bit wird gesetzt, wenn der Akkumulator null ist. (Alle Bits)
2	Interupt Disable Bit Wenn dieses Bit gesetzt ist, wird die CPU keine Unterbrechungsaufforderungen berücksichtigen. Die Unterbrechungsaufforderungen können per Hardware über die IRQ-Leitung erfolgen. Wenn das Bit auf Null zurückgesetzt wird, können Unterbrechungen wieder erkannt werden.

3 Dezimal-Betriebsart. Wenn dieses Bit gesetzt wird, werden die Übertrags-/Umlauf-Bits gesetzt, sobald das Ergebnis die Zahl 99 überschreitet. Wenn es auf Null zurückgesetzt wird, werden alle arithmetischen Operationen binär ausgeführt. (Übertrag wird dann erst beim Überschreiben von 255 ausgegeben).

4 BREAK-Command Bit. Dieses Bit wird gesetzt, während ein Interrupt abgearbeitet wird. Man kann daraus erkennen, ob der Interrupt durch den Breakbefehl oder durch ein Unterbrechungssignal ausgelöst wurde.

5 Dieses Bit ist unbenutzt und für spätere Entwicklungen vorgesehen.

6 Überlauf (Overflow). Dieses Bit zeigt an, daß eine binäre, arithmetische Operation durchgeführt wurde, die eine Überschreitung des Bitvolumens verursacht hat.

7 Negativ-Bit. Dieses Bit wird gesetzt, wenn das Ergebnis des zuletzt durchgeführten Befehls negativ ist. Bit 7 wird von allen arithmetischen, logischen Befehlen beeinflußt.
Wenn N="1" handelt es sich also um eine binäre, negative Zahl.
Dieses Bit eignet sich zur Realisierung von Funktionen, bei denen das äußere Bit links als Flag dienen kann, z. B. als Ein-/Ausgabe-Statusregister.

Akkumulator Der Akkumulator dient als universelle Speicherzelle. Operationen, welche sich auf zwei Speicherzellen beziehen, müssen zuerst über den Akkumulator laufen. Der Akkumulator ist eines der am meisten aktiv genutzten Register in der Maschine.

Die Indexregister X und Y

Diese Register dienen einmal als Zwischenspeicher für Teilergebnisse, weiterhin lassen sich ein paar arithmetische Operationen mit ihnen durchführen. Das X-Register kann weiterhin den Stackpointer laden bzw. lesen. Eine weitere wichtige Eigenschaft beider Indexregister liegt aber in ihrer Verwendung bei der indizierten und indirekten Adressierung.

Der Stack Pointer

Der Stack Pointer ist ein Register, in dem die Adresse enthalten ist, die zuletzt in den Stack abgelegt wurde. Der Stack ist ein Speicher mit LIFO-Eigenschaften (last in, first out).
Im 6502 kann der Stack maximal 256 Bytes enthalten. Der Stackpointer hat 8 Bit mit einer 01 davor. Dies legt den Stackbereich zwischen 0100 und 01FF fest.

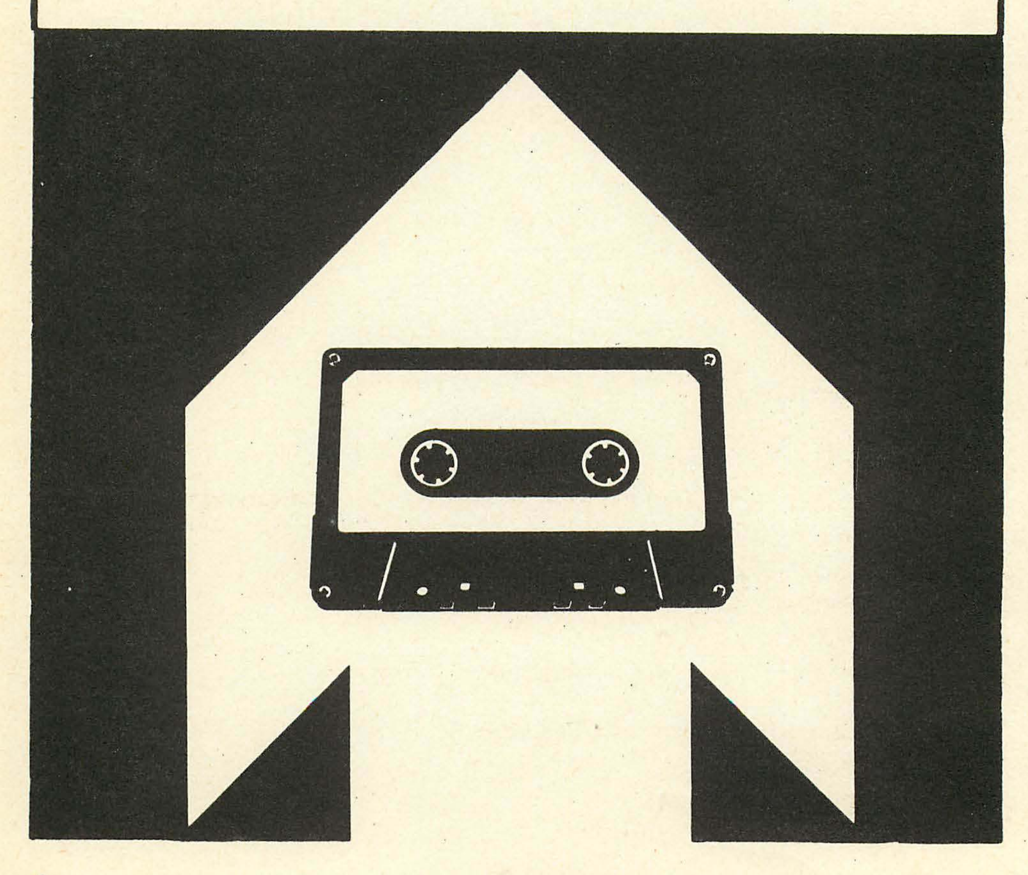
Beschreibung für Supermonitor Challenger Superboard

Nachfolgend geben wir Ihnen die Bedienungsanleitung für einen Monitor zur Programmierung in Maschinensprache auf dem Challenger Superboard. Der Monitor kann auf Cassette beim Fachhandel oder beim Hofacker Verlag direkt unter der Bestell-Nr.: 8183 für DM 49, – bezogen werden.

Für das Superboard ist weiterhin eine Cassette mit einer Einführung in die 6502 Maschinensprache lieferbar. Bestell-Nr.: 8182 DM 29,80

Bestellungen an den Buch- und Fachhandel oder direkt beim Ing. W. Hofacker GmbH Verlag.

Achtung: Cassetten sind vom Umtausch ausgeschlossen.



Extended Monitor für Superboard

Der im Superboard enthaltene Monitor ist recht einfach und für denjenigen, der in Maschinensprache programmieren möchte, doch nicht ausreichend.

Der nachfolgend beschriebene Supermonitor für das Challenger Superboard kann einfach mit dem L-Befehl im Monitor ab Adresse 0700 in den Speicher geladen werden.

Drücken Sie BREAK,

dann die M-Taste,

dann 700. Starten Sie den Cassettenrecorder und nach ca. 1 Umdrehung drücken Sie die L-Taste. Auf dem Bildschirm können Sie nun verfolgen, wie sich die Adressen erhöhen und die Daten eingelesen werden. Wenn die letzte Adresse 1002 Hex erreicht ist, bleibt diese Adresse mit dem letzten Inhalt auf dem Bildschirm stehen.

Jetzt drücken Sie wieder die BREAK-Taste und geben die Startadresse 800 ein. Anschließend drücken Sie bitte G. (nach BREAK wieder M)

Jetzt muß sich in der linken unteren Bildschirmecke ein Doppelpunkt mit Bindestrich zeigen: :–

Jetzt ist der Monitor gestartet und Sie können die nachfolgenden Kommandos geben.

Speicher-Inhaltsanzeige und Inhaltsänderung

1. Speicherzelle anzeigen und ändern
   Speicherbereich "ausdumpen"
   Speicherbereich füllen
   Speicherbereiche verschieben
   Speicherbereiche relocatieren

2. Fehlersuche in Programmen und Austesten von Programmen

Disassemblieren
Suchen nach einem Byte-String
Breakpoint setzen und Kontrolle
Registeranzeige des Prozessors
Starten von Programmen in Maschinensprache

3. Laden und Abspeichern mit Cassette

Laden von Cassette
Abspeichern auf Cassette
"View" – Eine Art "Verify"

4. Hexadezimale Arithmetic
Rechnen mit Hexadezimalzahlen
Anzeige des Überlaufanzeigers

Wie schon erwähnt, meldet sich der Monitor nach dem Start mit 800 G mit einem Doppelpunkt.
Der Monitor kann jetzt Kommandos entgegennehmen. Bei einer ungültigen Eingabe erfolgt der Ausdruck eines Fragezeichens.
Alle im folgenden aufgeführten Adressen und Daten verstehen sich in Hexadezimal.

Bedeutung einiger Kurzzeichen
LF = Line Feed
CR = Return-Taste
↑ = Shift / N auf dem Superboard

@ = Klammeraffe, Shift/P auf dem Superboard

Kommandos zur Anzeige des Speicherinhaltes:
@aaaa Zeigt Ihnen die Adresse und den Speicherinhalt der Zelle aaaa. Es können neue Daten eingegeben werden oder auch nicht. (Zwei Hex-Zahlen)
Darauf folgend können folgende Eingaben gemacht werden:
LF = Zeigt den Inhalt der nächsten Speicherzelle
↑ = Zeigt den Inhalt der vorhergehenden Speicherzelle
" = Zeigt den Inhalt der Speicherzelle als ASCII-Zeichen oder graphisches Symbol
CR = Rückkehr zum Kommando-Mode des Monitors

D bbbb, cccc dumped den Speicherinhalt von Adresse bbbb bis cccc-1 aus
F dddd, eeee = ff Füllt den Speicherbereich von Adresse dddd bis eeee-1 mit den Hexadezimalzahlen ff

M gggg = hhhh, iiii Bringt den Speicherinhalt zwischen Adresse hhhh und iiii an die Anfangsadresse gggg
Achtung: Beim Verschieben nach oben muß der Abstand größer sein, als der zu verschiebende Block!

Rgggg = hhhh, iiii Relocatiert den Speicherinhalt von Adresse hhhh bis iiii-1 an einen Speicherbereich, beginnend mit der Adresse gggg. Alle Dreibytebefehle des 6500 werden entsprechend geändert, die in den zu verschiebenden Bereich fallen.

Befehle zur Fehlersuche in Programmen (DEBUG)

Q NNNN

Dieser Befehl disassembliert den ausführbarenMaschinencode in Mnemomics. Die Disassemblierung startet an der hexadezimalen Adresse NNNN und läuft die 24 folgenden Zeilen weiter. Nicht ausführbarer Objectcode wird als Fragezeichen ausgegeben (???). Die Disassemblierung kann durch drucken der Line Feed Taste in weiteren 24 Zeilen-Blöcken fortgesetzt werden.

N HEX > VON, BIS

Dieser Befehl durchsucht den Speicherblock zwischen den Hexadezimaladressen VON und BIS nach dem Datenstring Hex. Wenn das gesuchte Hex-Wort gefunden ist, geht der Monitor in den Daten-MODE. (Datenmode = Shift P)

Die maximale Wortlänge des Strings, nach dem gesucht werden kann ist 8 Byte.

W ASCII > VON, BIS

Identisch Befehl N, jedoch wird hier nach einem ASCII-Zeichen und nicht nach einem Hex-Wort gesucht.

Bn, NNNN

Setzen eines Breakpoints "n" an der hexadezimalen Adresse NNNN. Bis zu acht Breakpoints (n = 1 bis n = 8) können gesucht werden.

Eu

Eliminieren des Breakpoints n.

T

Ausdrucken aller Breakpointadressen. Achtung! Die Adresse eines nichtspezifierten Breakpoints führt zu FFFF.

C

Continue. Fortsetzen des Programmablaufes vom letzten Breakpoint an. Kann nur nach Eingabe von Breakpoints verwendet werden.

I

Druckt die Adresse aus bei der Monitor zuletzt durch ein Break angehalten wurde. Der Inhalt der Register und die Stackpointer Adressen werden ausgegeben.

A; X; Y; P; K

Diese fünf Befehle drucken den Inhalt
des Akkumulators
des X-Registers
des Y-Registers
Status
und Stackpointer aus.
Die einzelnen Worte können geändert werden.

Laden und Speichern auf Audio-Cassette.

S aaaa, bbbb

Dieser Befehl lädt den Datenblock zwischen den Hexadezimaladressen aaaa und bbbb im folgenden "CHECKSUM FORMAT"
; LEN ADD DAT CHK

Wobei: ";" = Markierung des Anfanges eines Records
LEN = Länge des Records
ADD = Anfangsadresse des Records
DAT = Daten
CHK = Checksumme des Records

Die Ausgabe auf die Cassette bleibt solange erhalten, bis ein L-Befehl gegeben wird.

L

Lädt einen Datenblock von Cassette in den Speicher. Und zwar im Format wie oben angegeben. (Checksum)
Wenn ein Checksummenfehler erkannt wurde; wird ERR ausgedruckt. Halten Sie in diesen Moment den Cassettenrecorder an, spielen Sie etwas zurück. Unbedingt soweit, daß der Fehler passiert wird. Drücken Sie wieder die PLAY-Taste und geben L ein.

V

Mit diesem Befehl kann der Inhalt auf einer Cassette angesehen werden, ohne etwas in den Speicher zu laden.

G nnnn

Start der Programmausführung an der hexadezimalen Adresse nnnn.

HDT1, DT2 (+, - , *, /) = ANS

Dieser Befehl ermöglicht es ihnen das der Superboard als Rechner für 16 Bit Hexadezimalzahlen zu verwenden. DT1 und DT2 sind hierbei die beiden Hexadezimalzahlen, die Sie mit den Operatoren (+, - , *,/) verknüpfen können.
ANS ist das 16 Bit - Ergebnis

O

Gibt Ihnen ein Überlaufzeichen oder einen Reminder aus der oben beschriebenen 16 Bit. Multiplikation oder Division.

Datenblockverschiebung und Relokatieren.
Der Challenger Supermonitor beinhaltet noch zwei weitere wertvolle Kommandos.

MOVE und Relocate

Der "MOVE" Befehl verschiebt einen Block von Daten im Speicherbereich ohne daß irgend welche Daten verändert werden. Im Gegensatz

hierzu ändert der "Relocate" Befehl die Operanden so, daß sie in den neuen Speicherbereich passen.
Beispiel: Ein Programm zwischen $ 300 und $ 380 sieht wie folgt aus:

```
0300 A520     LDA $ 20
0302 206E03   JSR $ 036 E
036E 8DC6D0   STA $ D0C6
037F 60       RTS
```

Wenn wir den Befehl M0500=0300, 0380 verwenden, sieht der Code nach der Blockverschiebung wiefolgt aus:

```
0500 A520     LDA $ 20
0502 206E03   JSR $ 036E
056E 8DC6D0   STA $ D0C6
057F 60       RTS
```

Wenn wir auch noch den Befehl F 0300, 0380=00 noch angehängt hätten, würde das Programm auch nicht laufen.
Wenn wir den Befehl R 0500=0300, 0380 geben, erhalten wir das folgende Programm.

```
0500 A520     LDA $ 20
0502 206E05   JSR $ 056 E
056E 8DC6D0   STA $ D0C6
057F 60       RTS
```

Beachten Sie bitte, daß der Sprung ins Unterprogramm in Zeile 502 so abgeändert wurde, daß er in dem neuen Speicherbereich arbeiten kann. Wenn der Sprung jedoch an eine Adresse außerhalb des Bereiches 0300-0380 erfolgt wäre, wäre der Befehl nicht abgeändert worden.
Dies war der Fall in Zeile $ 56E.

Hier sollte noch erwähnt werden, daß nicht alle Programme direkt relocatiert werden können. Zum Beispiel ein Programm mit Datentabellen. Es könnte hier passieren, daß der Relocator einige Daten als ausführbaren Opcode interpretiert und so Fehler entstehen.

Bei Verwendung des MOVE- und Relocatier- Befehles muß man in jedem Falle darauf achten, daß die Entfernung für die Bewegung nach

vorne immer größer sein muß, als der zu bewegende Programmblock. Anderenfalls wird der Orginaldatenblock überschrieben.

Die Anwendung von Breakpoints bei der Fehlersuche in Programmen

Wie der Name schon sagt ist der Breakpoint ein Punkt im Programm, an dem ein laufendes Programm unterbrochen bzw. angehalten wird. Mit dem Superboard Supermonitor können Sie bis zu acht Breakpoints vor dem Programmstart in das Programm eingebaut werden. Wenn das Programm dann gestartet wird und an einen Breakpoint angekommen ist, kehrt das Superboard in den Monitor zurück und druckt die folgenden Informationen aus:

Bn @ NNNN
A/CC X/CC Y/CC P/CC K/CC

Wobei n die laufende Nummer der Breakpoints ist (1 - 8), NNNN ist die Hexadezimaladresse an der der Breakpoint angetroffen wurde und der Rest sind die Registerinhalte zur Zeit des Breakpoints. Bevor das Programm wieder gestartet werden soll, können diese Register verändert werden.

Um die Anwendung etwas zu erläutern sehen Sie sich bitte einmal den nachfolgenden Programmteil an.

```
0350 B003     BCS $ 0355
0352 4C8003   JMP $ 0380
0355 20 0050  JSR $ 0500
```

Wenn an Adresse $ 350 über den Befehl B1, 0350 jetzt ein Breakpoint gesetzt wird, so wird der OPCODE BO entfernt und vom Supermonitor abgespeichert (gerettet). Anstelle von VO wird der Opcode des Break-Befehls 00 an Adresse $ 350 eingesetzt. Wenn das Programm dann gestartet wird und am Breakpoint 1 ankommt, springt das Superboard wieder in den Monitor und zeigt folgendes an:

B1 @ 0350
A/00 X/00 Y/45 P/35 K/FC

Zusätzlich wird Breakpoint Nr. 1 ausgelöscht und der Befehl BO wieder eingesetzt. Beim Einsehen in das Statusregister sehen wir, daß das Carry-Flag gesetzt ist. Das Programm würde dann nach Adresse $ 355 springen und dort den Befehl JSR $ 0500 ausführen.

Wenn es sich herausgestellt hat, daß das Programm besser arbeitet, wenn nach Adresse $ 380 gesprungen wird (JMP $ 0380, so braucht der Programmierer nur P einzugeben und ändert dann den Inhalt des Status-Register auf $ 34 wobei das Carry-Flag gelöscht wird. Wenn nun der Befehl zum Weiterfahren im Programm gegeben wird (Continue) wird die Programm-Verzweigung nicht erfolgen.

Die Breakpointadressen und die Zwischengespeicherten Opcodes werden in der Zero-Page im Bereich $ E8 - $ FF abgelegt. Achtung! Diese Zellen sollten deshalb während der Fehlersuche nicht überschrieben werden.

Zusammenfassung der Kommandos und Befehle

Kommando	Funktion	Bemerkung
@ NNNN	Öffnen der Speicherzelle NNNN	"Line Feed"=ermöglicht Weiterschaltung auf die nächste Adresse. "↑"=zeigt auf die vorhergehende Speicherzelle. "=druckt den Adressinhalt in ASCII aus. /=Zurückkehr in Data Mode

A	Druckt Breakpoint Akkumulator aus.	Kehrt automatisch in den DATA-Mode zurück
B	Eingabe der Breakpoints (n)	n=1 bis 8
C	Continue (Weiterarbeiten vom letzten Breakpoint an.	Monitor muß auch über BREAK gestoppt worden sein.
D	Dumpausgabe aus dem Speicher VON, BIS (bbbb, eeee)	
E	Eliminiere Breakpoint n	
F	Fülle den Speicher im Bereich VON, BIS (dddd, eeee)	
G	Starte das Programm an Adresse NNNN	
H	Initialisierung des Hexadezimalrechners	Vier Funktionen sind möglich (+,- ,*,/)
N	Suchen nach einem Zahlen-String	8 Byte Max.
I	Drucke Adresse der letzten Breakpointeingabe	
K	Drucke den Breakpoint Stackpointer aus	Führt zurück in den Daten-Mode
L	Laden des Speichers von der Cassette	
M	Verschieben eines Speicherblockes NEU=VON, BIS	
O	Zeigt den Übertrag bei einem Überlauf. (Im Hexadezimalrechner-Mode)	
P	Druckt das Breakpoint Statusregister aus	Geht automatisch in den Data-Mode
Q	Disassemblieren von NNNN angefangen	Mit Line-Feed folgt die nächste Bildschirmfüllung
R	Relokatiere den Code Neu = VON, BIS	
S	Save: Speichern des Speicherbereiches VON, BIS	
T	Drucke Breakpoint Adresstabelle	
V	Ansehen des Cassetteninhaltes auf dem Bildschirm.	

W	Suchen nach Wort-String in ASCII VON, BIS	8 Byte max.
X	Drucke Breakpoint X-Register aus	geht in Data-Mode
Y	Drucke Breakpoint Y-Register aus	geht in Data-Mode

Nützliche Unterprogramme:

INCH	Eingabe in Akku mit Echo	$ 853
OUTCH	Ausgabe eines Zeichens vom Akku	$ 861
PRBYT	Gibt das Byte im Akku aus	$ AAC
CRLF	Ausgabe Carriage Return und Line-Feed	$ 807

Der Supermonitor kann auf Cassette vom Hofacker-Verlag oder durch den Fachhandel under der Best.-Nr. 8183 für DM 49,-- bezogen werden.

6502 OP CODES

Arranged in logical order by Jim Butterfield, Toronto

	IMM 2	ZPAG 2	Z,X 2	Z,Y 2	ABS 3	A,X 3	A,Y 3
ASL		06	16		0E	1E	
ROL		26	36		2E	3E	
LSR		46	56		4E	5E	
ROR		66	76		6E	7E	
STX		86		96	8E		
LDX	A2	A6		B6	AE		BE
DEC		C6	D6		CE	DE	
INC		E6	F6		EE	FE	

Op Code ends in - 2, - 6, or - E

	IMM 2	ZPAG 2	Z,X 2	ABS 3	A,X 3
BIT		24		2C	
STY		84	94	8C	
LDY	A0	A4	B4	AC	BC
CPY	C0	C4		CC	
CPX	E0	E4		EC	

Misc. - 0, - 4, - C

BPL	10	BMI	30
BVC	50	BVS	70
BCC	90	BCS	B0
BNE	D0	BEQ	F0

Branches - 0

	ABS	(IND)
JSR	20	
JMP	4C	6C

Jumps

	IMM 2	ZPAG 2	Z,X 2	(I,X) 2	(I),Y 2	ABS 3	A,X 3	A,Y 3
ORA	09	05	15	01	11	0D	1D	19
AND	29	25	35	21	31	2D	3D	39
EOR	49	45	55	41	51	4D	5D	59
ADC	69	65	75	61	71	6D	7D	79
STA		85	95	81	91	8D	9D	99
LDA	A9	A5	B5	A1	B1	AD	BD	B9
CMP	C9	C5	D5	C1	D1	CD	DD	D9
SBC	E9	E5	F5	E1	F1	ED	FD	F9

Op Code ends in - 1, - 5, - 9, or - D

	0–	1–	2–	3–	4 –	5–	6 –	7–
-0	BRK				RTI		RTS	
-8	PHP	CLC	PLP	SEC	PHA	CLI	PLA	SEI
-A	ASL-A		ROL-A		LSR-A		ROR-A	

	8–	9 –	A–	B–	C–	D–	E –	F –
-0								
-8	DEY	TYA	TAY	CLV	INY	CLD	INX	SED
-A	TXA	TXS	TAX	TSX	DEX		NOP	

Single-byte Op Codes - 0, - 8, - A

Another OP-CODE chart? Yes, but there is a reason.

This chart groups the codes logically. This way, you get three benefits.

First, you get to see how the codes are classified and decoded. A glance at the chart shows that LDA and ADC, for example, are close cousins:

same addressing modes, same timing, and quite similar OP-CODES; on the other hand, LDA and LDX are noticeably different. The classification idea can be useful to those who want to dig into op-codes,say to write an assembler or a disassembler.

Secondly, ist's handy for looking up an OP-Code-maybe easier than an alphabetical list. You"ll very quickly learn to look at the right box and spot the code you want right away. As you get used to the groupings, you'll also develop a feel for the addressing modes that are allowed.

Thirdly, you"ll find it convenient for identifying an unknown op-code-- ("What the heck is CE, anyway?")

Jim B.

Editors Note: I have found this chart to be extremely useful in designing opcode decode algorithms etc.

Aus 6502 User Notes von:

Eric C. Rehnke
Mail Stop RC 55
P.O. Box 3669
Anaheim, California 92803
U.S.A.

Befehlsliste und entsprechende Befehlserklärung für 65XX - Prozessor

Sicher ist es Ihnen schon einmal passiert, daß Sie ein Assembler-Listing gesehen haben und nur zum Teil die Befehle deuten konnten. Der größte Teil lag also im Dunkeln. Das einfache Eingeben der Befehle ist einfach, aber zum Verständnis des Programmablaufes muß man die genaue Bedeutung der Befehle kennen.

Aus diesem Grunde versuche ich in einfachster Weise mit jeweils einem entsprechenden Beispiel, Ihnen die Befehlsbedeutung zu erklären.

Fangen wir an: XX beliebige Hex-Zahl



A9 XX
Es wird das Folgebyte in den Akkumulator geladen

z.B.	A9 AA	Die Hex-Zahl AA wird in Akku geladen

AD XX XX
Es wird der Inhalt der Speicherzelle in den Akkumulator geladen, der sich aus den Folgebytes ergibt. Zu beachten wäre Lower- und Higherbyte.

z.B.	AD 00 17	Der Inhalt der Adresse 1700 wird in den Akku geladen.

A5 XX
Es wird der Inhalt der Speicherzelle in den Akkumulator geladen, die sich aus dem Folgebyte ergibt. Es ist eine Adresse in der Zeropage.

z.B.	A5 0A	der Inhalt der Adresse 00 0A wird in den Akku geladen.

A1 XX
Es wird der Inhalt des X-Register's zum Folgebyte addiert. Das Ergebnis dieser Addition ist eine Adresse in der Zeropage. Deren Inhalt ist das Lowerbyte und die nächstfolgende Adresse ist das Higherbyte. Der Inhalt der Adresse die sich aus Lower- und Higherbyte ergibt wird in den Akkumulator geladen.

z.B.	A1 03	X-Registerinhalt ist z.B. 04; 03 + 04 → 07 Inhalt der Adresse 0007 ist z.B. AA Inhalt der nächstfolgenden Adresse, also 0008 ist z.B. 01 Lowerbyte ist AA; Higerbyte ist 01. Es wird also der Inhalt der Adresse 01 AA in den Akku gespeichert.

wichtig: Es wird kein Überlauf bei der Addition des X-Registerinhalts und des Folgebytes geprüft. d.h. FE + 04 ergibt nicht 01 02 sondern 02. Die 1 wird nicht berücksichtigt.

B1 XX

Es wird der Inhalt der Adresse des Folgebytes zum Y-Registerinhalt dazu addiert. Dies ergibt eine neue Adresse, deren Inhalt in den Akkumulator geladen wird.

Tritt bei der Addition zu, Folgebyte und Y-Registerinhalt ein Überlauf auf (Carry-Flag wird gesetzt) so wird der Überlauf zu der nächstfolgenden Adresse die im Folgebyte angegeben ist addiert. Dabei ist nun der Wert ohne Überlauf das Lowerbyt und die Addition, Überlauf und Inhalt der nächstfolgenden Adresse, das Higherbyt. Tritt hier ein Überlauf auf, so wird er nicht berücksichtigt. Der Inhalt der Adresse die sich aus Lower- und Higherbyte ergibt wird in den Akkumulator geladen.

1. ohne Überlauf

z.B. B1 07 — Y-Registerinhalt ist z.B. 04
Inhalt der Speicherzelle 0007 ist z.B. 01
01 + 04 → 05
der Inhalt der Speicherzelle 0005 wird in den Akku geladen.

2. mit Überlauf

z.B. B1 07 — Y-Registerinhalt ist z.B. 04
Inhalt der Speicherzelle 0007 ist z.B. FE
FE + 04 → 1 Überlauf 02 d.h. Überlauf hat stattgefunden (Carry-Flag ist gesetzt). 2 ist nun das Lowerbyte und die Adition des Zelleninhaltes der Adresse Folgebyte 07 + 1 also Adresse 0008 und Überlauf 1 ist das Higherbyte.
z.B. Inhalt der Adresse 0008 ist 04
04 + Überlauf ist nun das Higherbyte.
Lower- und Higherbyte ergeben nun die Adresse 0502. Der Inhalt dieser Adresse wird in den Akku geladen.

Bemerkung:

Ist zum Beispiel der Inhalt der Adresse 0008 = FF wobei die Addition mit dem Überlauf wieder einen Überlauf erzeugen würde, so wird diesmal der neuerlich erzeugte Überlauf nicht berücksichtigt.
Also FF + 1 → 1 00
Hier würde der Inhalt der Adresse 0002 in den Akku geladen.

B5 XX

Es wird das Folgebyte mit dem Inhalt des X-Register's addiert. Die Addition ist eine neue Adresse, deren Inhalt in den Akkumulator geladen wird.

z.B.	B5 07	X-Registerinhalt ist z.B. 01; 07 + 01 → 08 Inhalt der Adresse 0008 ist z.B. FF, wobei FF nun in den Akku geladen wird.

BD XX XX

Es werden die Adresse, die sich aus den 2 Folgebytes ergibt, Lower- und Higherbyte berücksichtigt, und der Inhalt des X-Register's addiert. Die Addition ergibt eine neue Adresse, deren Inhalt in den Akkumulator geladen wird.

z.B.	BD 00 17	X-Registerinhalt ist z.B. 01; 1700 + 01 → 1701 Inhalt der Adresse 1701 wird in Akku geladen.

B9 XX XX

Ist der analoge Befehl zu BD XX XX, nur im Unterschied dazu wird hier das Y-Register angesprochen.



A2 XX

Es wird das Folgebyte in das X-Register geladen.

z.B. A2 FO die Hex Zahl FO wird in X-Register geladen.

AE XX XX

Es wird der Inhalt der Speicherzelle geladen, die sich aus den Folgebytes ergibt. Lower- und Higherbytes berücksichtigt.

z.B. AE 00 10 der Inhalt der Adresse 1000 wird in X-Register geladen.

A6 XX

Es wird der Inhalt der Speicherzelle in das X-Register geladen, die sich aus dem Folgebyte ergibt. Es ist eine Adresse in der Zeropage.

z.B. AG 10 Der Inhalt der Adresse 0010 wird in X-Register geladen.

B6 XX

Es wird das Folgebyte mit dem Inhalt des Y-Register's addiert. Die Addition ist eine neue Adresse, deren Inhalt in das X-Register geladen wird.

z.B. B6 08 Y-Registerinhalt sei 01; 08 + 01 → 09
Inhalt der Adresse 09 wird in das X-Register geladen.

BE XX XX

Es werden die Adresse, die sich aus dem 2 Folgebytes ergibt. Lower- und Higherbyt berücksichtigt, und der Inhalt des Y-Register's addiert. Die Addition ergibt eine neue Adresse deren Inhalt in das X-Register geladen wird.

z.B. BE 00 09 Y-Registerinhalt sei z.B. 01; 0900 + 01 → 0901
Inhalt der Adresse 0901 wird im X-Register geladen.

LDY

A0 XX

Es wird das Folgebyte in das Y-Register geladen.

z.B.	A0 EE	die Hex-Zahl EE wird ins Y-Register geladen.

AC XX XX

Es wird der Inhalt der Speicherzelle, die durch die Folgebytes angegeben wird, Lower- und Higherbytes berücksichtigt, in das Y-Register geladen.

z.B.	AC 00 08	Der Inhalt der Adresse 0800 wird ins Y-Register geladen.

A4 XX

Es wird der Inhalt der Speicherzelle, die durch das Folgebyte angegeben ist in das Y-Register geladen.

z.B.	A4 0F	Der Inhalt der Adresse 0F wird in Y-Register geladen.

B4 XX

Es wird das Folgebyte mit dem Inhalt des X-Register's addiert. Die Addition ist die neue Adresse, die in das Y-Register geladen wird.

z.B.	B4 0F	X-Registerinhalt sei z.B. 02; 0F + 02 → 11 Inhalt der Adresse 11 wird ins Y-Register geladen.

BC XX XX

Es werden die Adresse, die sich aus den 2 Folgebytes ergibt, Lower- und Higherbyte berücksichtigt, und der Inhalt des X-Registert addiert. Das Y-Register wird mit dem Inhalt der aus Addition ergebenen Adresse geladen.

z.B.	BC 00 06	X-Registerinhalt sei z.B. 04; 0600 + 04 → 0604 Inhalt der Adresse 0604 wird ins Y-Register geladen.

STA

8D XX XX

Der Inhalt des Akkumulator's wird in die Adresse gespeichert, die sich aus den Folgebytes ergeben, Lower und Higherbyte berücksichtigt.

z.B. 8D 00 05 Akkuinhalt sei z.B. DD
DD wird in die Adresse 0500 gespeichert

85 XX

Der Inhalt des Akkumulators wird in die Adresse gespeichert, die sich aus dem Folgebyte ergibt. Die Adresse liegt in der Zeropage.

z.B. 85 07 Akkuinhalt sei z.B. 01
01 wird in die Adresse 07 gespeichert.

81 XX

Es wird der Inhalt des X-Register's zum Folgebyte addiert. Das Ergebnis dieser Addition ist eine Adresse in der Zeropage. Deren Inhalt ist das Lowerbyte. Die nächstfolgende Adresse ist das Higherbyte. Der Akkuinhalt wird in die Adresse gespeichert, die sich aus Lower- und Higherbyte ergibt.

z.B. 81 07 X-Registerinhalt sei z.B. 04
07 + 04 → 0B
Inhalt der Adresse 000B sei z.B. AB
Inhalt der nächstfolgenden Adresse also 000C, sei z.B. 02
Lowerbyte = AB; Higherbyte = 02
Es wird der Akkuinhalt in die Adresse 02AB gespeichert.

Bemerkung:

Tritt ein Überlauf bei der Addition des X-Registerinhaltes und dem Folgebyte auf, so wird der Überlauf nicht berücksichtigt. d.h. FF + 02 ergibt 101 wobei nur 01 bewertet wird.

91 XX

Es wird der Inhalt der Adresse des Folgebytes zum Y-Registerinhalt dazu addiert. Diese Addition ergibt eine neue Adresse, in die der Akkuinhalt gespeichert wird.

Tritt bei der Addition zu Folgebyte und Y-Registerinhalt ein Überlauf auf (Carry-Flag wird gesetzt), so wird der Überlauf zu der nächstfolgenden Adresse die im Folgebyte angegeben ist addiert. Der Wert ohne Überlauf ist das Lowerbyte und die Addition Überlauf nächstfolgender Adresseninhalt das Higherbyte. Tritt hier ein Überlauf auf, so wird er nicht berücksichtigt. Der Akkuinhalt wird in die Adresse gespeichert, die sich aus Lower- und Higherbyte ergibt.

Beispiel

ohne Überlauf:

91 08 Y-Registerinhalt 05
Inhalt der Speicherzelle 0008 ist z.B. 04
04 + 05 → 09
Akkuinhalt wird in Speicherzelle 0009 geladen

mit Überlauf:

91 08 Y-Registerinhalt 05
Inhalt der Speicherzelle 0008 ist z.B. FD
FD + 05 → Überlauf 1 02
Ein Überlauf hat stattgefunden.
02 ist nun das Lowerbyte
Die Addition des Zelleninhaltes der Adresse Folgebyte + 1 (08 + 1) also Inhalt der Adresse 09 und Überlauf 1 ergeben das Higherbyte z.B. Inhalt der Adresse 0009 ist 04. 04 + 1 (Überlauf) ist das Higherbyte also 05 Lower- und Higherbyte ergeben die Adresse 0502. Der Akkuinhalt wird in diese Adresse geladen.

Bemerkung:

Ist z.B. der Inhalt der Adresse 0009 = FF wobei die Addition mit dem vorherigen Überlauf wieder einen Überlauf ergibt, so wird der neuerliche Überlauf nicht berücksichtigt.
also FF + 1 → 1 00
d.h. 00 ist das Higherbyte. Der Inhalt des Akku würde also in Adresse 00 02 geladen.

95 XX
Es wird das Folgebyte mit dem Inhalt des X-Register's addiert. Das Ergebnis ist eine neue Adresse in die der Inhalt des Akkumulator's geladen wird.

z.B.	95 09	X-Registerinhalt sei z.B. 01; 09 + 01 → 0A Akkuinhalt wird in Adresse 000A gespeichert.

9D XX XX
Es wird die Adresse die sich aus den beiden Folgebytes ergibt (Lower- und Higherbyte) und der Inhalt des X-Register's addiert. Das Ergebnis der Addition ist die Adresse, in die der Akkuinhalt gespeichert wird.

z.B.	9D 00 01	X-Registerinhalt sei z.B. 01; 0100 + 01 → 0101 Inhalt des Akku wird in Adresse 0101 geladen.

99 XX XX
Dies ist der analoge Befehl zu 9D XX XX, nur daß hier mit dem Y-Register gearbeitet wird.

STX

8E XX XX

Es wird der Inhalt des X-Registers in die Speicherzelle geladen, die sich aus den beiden Folgebytes ergibt (Lower- und Higherbyte berücksichtigt)

z.B.	8E 00 01	X-Registerinhalt wird in Speicherzelle 0100 gespeichert.

86 XX

Es wird der Inhalt des X-Registers in die Speicherzelle geladen, die durch das Folgebyte angegeben ist.
Es ist eine Adresse aus der Zeropage

z.B.	86 0F	X-Registerinhalt wird in der Adresse 000F abgespeichert.

96 XX

Es wird das Folgebyte mit dem Inhalt des Y-Registers addiert. Das Ergebnis ist die Adresse in das der X-Registerinhalt gespeichert wird.

z.B.	96 F0	Y-Registerinhalt soll z.B. 01 sein F0 + 01 → F1 Inhalt des X-Registers wird in F1 gespeichert. Kommt bei der Addition ein Überlauf zustande so wird er nicht verarbeitet z.B. F1 + 0F → 100 hier wird X-Registerinhalt in 0000 gespeichert.

STY

8C XX XX

Es wird der Inhalt des Y-Registers in die Adresse geladen, die sich aus den Folgebytes ergeben (Lower- und Higherbyte berücksichtigt).

z.B.	8C 00 03	d.h. Y-Registerinhalt wird in Adresse 0300 abgespeichert.

84 XX

Es wird der Inhalt des Y-Registers in die Adresse geladen die sich aus den Folgebyte ergibt.

z.B.	84 01	d.h. Y-Registerinhalt wird in Adresse 0001 gespeichert.

94 XX

Es wird das Folgebyte mit dem Inhalt des X-Registers addiert. Das Ergebnis ist die Adresse, in die das Y-Register gespeichert wird.

z.B.	94 FF	X-Registerinhalt sei 02 FF + 02 → Überlauf 1 01 Y-Registerinhalt wird in Speicherzelle 0001 abgespeichert. Der Überlauf wird nicht verarbeitet. Er geht verloren.

TAX

AA

Der Inhalt des Akkumulator's wird in das X-Register geladen.

z.B. Akkuinhalt ist FF

Nach Ausführung des Befehls steht FF im X-Register.

TAY

A8

Der Inhalt des Akkumulator's wird in das Y-Register geladen.

z.B. Akkuinhalt ist 11

Nach Ausführung des Befehls steht 11 im Y-Register.

TSX

BA

Der Stackpointerinhalt wird in das X-Register geladen.

z.B. Stackpointerinhalt ist 02

Nach Ausführung des Befehls steht 02 im X-Register.

TXA

8A
X-Registerinhalt wird in Akku geladen.

z.B. X-Registerinhalt ist FA
Nach der Ausführung des Befehls steht FA im Akku.

TXS

9A
X-Registerinhalt wird in Stackpointeradresse gespeichert.

z.B. X-Registerinhalt 03
Nach der Ausführung steht 03 in Stackpointer-adresse.

TYA

98
Y-Registerinhalt wird in Akku abgespeichert.

z.B. Y-Registerinhalt 09
Nach der Ausführung steht 09 im Akku.

69 XX

A + M + C → A

Es wird das Folgebyte und der momentane Akkuinhalt addiert und in Akku abgespeichert. Tritt ein Überlauf auf, so wird der Überlauf mit addiert wenn vorher Carry-Flag gesetzt war, sonst nicht.

z.B. 69 05 Akkuinhalt ist FE
FE + 05 → 1 Überlauf 03 + Überlauf 01 und wenn vorher Carry-Flag gesetzt → 04
d.h. 04 wird im Akku abgespeichert.

6D XX XX

Es wird der Inhalt der Adresse die aus den beiden Folgenbytes hervor gehen mit dem momentanen Akkuinhalt addiert.
Tritt dabei ein Überlauf auf so wird dieser dazu addiert, wenn vorher das Carry-Flag gesetzt war; sonst nicht.

z.B. 6D 00 01 Akkuinhalt 0F
Inhalt der Zelle 0100 soll 03 sein
d.h. 0F + 03 = 12 wird in Akku abgespeichert.

65 XX

Es wird der Inhalt der Adresse, die aus dem Folgebyte hervorgeht mit dem momentanen Akkuinhalt addiert.
Tritt dabei ein Überlauf auf, wird er addiert wenn vorher das Carry-Flag gesetzt war. War es vorher nicht gesetzt so wird der Überlauf nicht addiert sondern nur das Carry-Flag gesetzt.

z.B. 65 03 Akkuinhalt = 04
Inhalt der Zelle 0003 sei 01
d.h. 01 + 04 → 05 wird in Akku geladen.

61 XX

Es wird der Inhalt des X-Registers zum Folgebyte addiert. Das Ergebnis dieser Addition ist eine Adresse in der Zeropage. Deren Inhalt ist das Lowerbyte und die nächstfolgende Adresse ist das Higherbyte. Der Inhalt der sich aus Lower- und Higherbyte ergibt und der momentane Akkuinhalt wird in den Akku geladen.

z.B. 61 03 X-Registerinhalt ist z.B. 04
momentaner Akkuinhalt ist 06
03 + 04 → 07; Inhalt der Adresse 0007 ist z.B. AA.
Inhalt der nächstfolgenden Adresse, also 0008, ist z.B. 01
Es wird also der Inhalt von der Adresse 01 AA zum momentanen Akkuinhalt dazuaddiert und im Akku gespeichert.

Bemerkung:

Tritt bei Addition des X-Registerinhaltes und des Folgebytes ein Überlauf auf so wird er dazugezählt wenn vorher das Carry-Flay gesetzt war. War es nicht gesetzt so wird der Überlauf vernachlässigt und gleichzeitig wird das Carry-Flag gesetzt.

71 XX

Es wird der Inhalt der Adresse des Folgebytes zum Y-Registerinhalt dazu addiert. Dies ergibt eine neue Adresse deren Inhalt mit dem momentanen Inhalt des Akku addiert wird und in den Akku gespeichert wird. War vorher noch das Carry-Flag gesetzt so wird auch dieses subtrahiert. Tritt nun ein Überlauf in der Addition des Folgebytes mit dem Y-Registerinhalt auf, so wird der Überlauf zu der nächstfolgenden Adresse die im Folgebyte angegeben ist addiert. Dabei ist nun der Wert ohne Überlauf das Lowerbyte und die Addition Überlauf und Inhalt der nächstfolgenden Adresse das Higherbyte. Tritt hier ein Überlauf auf, so wird er nicht berücksichtigt. Der Inhalt der Adresse die sich aus Lower- und Higherbyte ergibt wird mit dem momentanen Akkumulatorinhalt addiert. Das Carry-Flag wird vom Ergebnis auch noch subtrahiert.

ohne Überlauf:

z.B. 71 07 Y-Registerinhalt ist z.B. 04
Inhalt der Speicherzelle 0007 ist z.B. 01
01 + 04 → 05
Der Inhalt der Speicherzelle 0005 wird zum momentanen Akkuinhalt addiert. Inhalt sei z.B. 05
angenommen momentaner Akkuinhalt sei 0A und Carry-Flag sei 0.
Dann steht nach Befehlsausführung 0F im Akku.

mit Überlauf:

z.B. 71 07 Y-Registerinhalt sei z.B. 04
Inhalt der Speicherzelle 0007 ist z.B. FE
FE + 04 → 1 (Überlauf) 02
03 ist nun das Lowerbyte
Die Adresse Folgebyte +1 also 0008 ist z.B. 04
Der Überlauf dazu addiert ergibt: 04 + 1 → 05
05 ist nun das Higherbyte
Lower- und Higherbyte ergeben nun die Adresse 0502. Der Inhalt dieser Adresse wird mit dem Akkuinhalt addiert.
z.B. momentaner Akkuinhalt sei z.B. 0F
Inhalt der Adresse 0502 z.B. 03
Nach Befehlsausführung wird 12 als Akkuinhalt stehen. War vor der Befehlsausführung das Carry-Flag gesetzt so steht nach der Befehlsausführung 13 im Akku.

75 XX

Es wird das Folgebyte mit dem Inhalt des X-Register's addiert. Die Addition ist eine neue Adresse, deren Inhalt mit dem Akkuinhalt addiert wird. War vor der Befehlsausführung das Carry-Flag gesetzt, so wird auch dieses noch subtrahiert.

z.B. 75 07 X-Registerinhalt sei z.B. 02
07 + 02 → 09
Inhalt der Adresse 0009 sei z.B. FE
momentaner Akkuinhalt soll z.B. 03 sein. Carry-Flag soll vorher schon gesetzt sein.
Nach der Befehlsausführung wird FE + 03 + 01 also 02 im Akku stehen.
Der Überlauf wird nicht berücksichtigt in der Rechnung. Das Carry-Flag bleibt daher gesetzt.

7D XX XX

Es wird die Adresse, die sich aus den 2 Folgebytes ergibt (Lower- und Higherbyte) und der Inhalt des X-Register's addiert. Dieses Additionsergebnis ergibt eine neue Adresse, deren Inhalt mit dem Akkumulator addiert wird. Ein eventuell vorher gesetztes Carry-Flag wird auch noch addiert.

z.B. 7D 01 03 X-Registerinhalt z.B. 02
0301 + 02 → 0303
Inhalt der Adresse 0303 sei z.B. 3F
Inhalt des Akku sei z.B. 05
Carry-Flag sei 0.
Nach der Befehlsausführung wird 44 im Akku stehen.

79 XX XX
Ist der analoge Befehl zu 7D XX XX hier wird nur das Y-Register angesprochen.

29 XX

UND FUNKTION

A	B	Y
L	L	L
L	H	L
H	L	L
H	H	H

Es wird das Folgebyte mit dem Akkumulatorinhalt logisch UND verknüpft.
Das Ergebnis steht dann im Akku.

z.B. 29 FE Akkuinhalt sei z.B. 81

1	1	1	1	1	1	1	0
1	0	0	0	0	0	0	1

1 0 0 0 0 0 0 0

Nach Befehlsausführung wird im Akku 80 als Inhalt sein.

2D XX XX

Der Inhalt der Speicherzelle, die sich aus den beiden Folgebytes ergibt (Lower-und Higherbyte) wird mit dem Inhalt des Akkumulator's logisch UND verknüpft. Das Ergebnis steht nach Befehlsausführung im Akku.

z.B. 2D 01 02 Der Inhalt dieser Speicherzelle 0201 sei 01 angenommen, der Akkuinhalt soll 02 sein.
Nach der Logisch UND Verknüpfung wird 00 im Akku stehen.

25 XX

Der Inhalt der Speicherzelle, die durch das Folgebyte angegeben ist (Zeropageadresse) wird mit Akkumulatorinhalt logisch UND verknüpft. Das Ergebnis steht dann im Akku.

z.B. 25 04 Akkuinhalt sei FF
Inhalt der Speicherzelle 0004 sei 01
Nach der Ausführung des Befehls steht 01 im Akku.

21 XX

Es wird der Inhalt des X-Registers zum Folgebyte addiert. Das Additionsergebnis ist eine Adresse in der Zeropage. Deren Inhalt ist das Lowerbyt und der nächstfolgende Adresseninhalt ist das Higherbyte. Der Inhalt der aus Lower- und Higherbyte ergebenden Adresse wird mit dem Akkumulatorinhalt logisch UND verknüpft. Ein eventueller Überlauf bei der Addition wird nicht berücksichtigt.

z.B. 21 05 X-Registerinhalt sei 08
05 + 08 → 0D
Inhalt der Adresse 0D sei z.B. 03
Inhalt der Adresse 0E sei z.B. 02
Akkuinhalt sei z.B. 01
Inhalt der Adresse 0203 sei 02
Nach Einführung des Befehls steht im Akku 00.

31 XX

Es wird der Inhalt der Adresse des Folgebytes zum Y-Register addiert. DasAdditionsergebnisist eine Adresse deren Inhalt mit dem Akkuinhalt logisch UND verknüpft wird. Das Ergebnis steht anschließendim Akku. Tritt bei der Addition ein Überlauf auf d.h. das Carry-Flag wird gesetzt, so wird der Überläufer zu der nächstfolgenden Adresse die im Folgebyte angegeben ist, addiert. Dabei ist nun der Inhalt der Folgebyteadresse das Lowerbyte und der Inhalt der nachfolgenden Adresse (Überlauf dazu addiert) das Higherbyte.

Tritt ein Überlauf zwischen Überlauf und Inhalt der nächstfolgenden Adresse auf, so wird der Überlauf nicht berücksichtigt. Der Inhalt der aus Lower- und Higherbyte bestimmten Adresse wird mit dem Akkumulatorinhalt log. UND verknüpft. Das Ergebnis wird in den Akkumulator geladen.

ohne Überlauf:

z.B. 31 07 Y-Registerinhalt ist z.B. 05
Inhalt der Speicherzelle 07 ist z.B. 01
01 + 05 → 06
Inhalt des Akku's soll z.B. 09 sein.
Der Inhalt der Speicherzelle 0006 sei z.B. 07
Nach der Ausführung des Befehls steht im Akku als Inhalt 01

mit Überlauf:

z.B. 31 07 Y-Registerinhalt ist z.B. 05
Inhalt der Speicherzelle 07 ist z.B. FE
FE + 05 → 1, 03 d.h. Überlauf hat stattgefunden
Überlauf

(Carry-Flag ist gesetzt). 03 ist nun das Lowerbyte und die Addition des Zelleninhaltes der Folgebyteadresse und dem Überlauf ist das Higherbyte.
z.B. Inhalt der Adresse 0008 ist 04.
04 und Überlauf (1) ist nun das Higherbyte. Lower- und Higherbyte ergeben nun die Adresse 0503. Der Inhalt dieser Adresse wird mit dem Akkumulatorinhalt logisch UND verknüpft.

Bemerkung:

Ist z.B. der Inhalt der Adresse 0008 = FF so ergibt die Addition mit dem Überlauf wieder einen Überlauf. Doch der neuerlich erzeugte Überlauf wird nicht berücksichtigt. Also FF + 1 → 100. Der Inhalt der Zelle 0003 würde logisch UND verknüpft.

35 XX

Es wird das Folgebyte mit dem Inhalt des X-Registers addiert. Die Addition ist eine neue Adresse, deren Inhalt mit dem Akkumulatorinhalt logisch UND verknüpft wird.

z.B. 35 09 X-Registerinhalt ist z.B. 04
Akkuinhalt sei 01
09 + 04 → 0D
Inhalt der Adresse 0D sei z.B. 0002
Nach der Ausführung wird im Akkumulator 00 stehen.

3D XX XX

Es wird die Adresse, die sich aus den beiden Folgebytes ergibt (Lower- und Higherbyte) mit dem Inhalt des X-Register's addiert. Dieses Additionsergebnis ergibt eine neue Adresse, deren Inhalt mit dem Akkumulatoinhalt logisch UND verknüpft wird. Das Ergebnis steht im

Akku.

z.B. 35 00 03 X-Registerinhalt ist z.B. 03
0300 + 03 → 0303
Inhalt der Adresse 0303 sei z.B. 01
Nach der Ausführung wird 00 im Akkumulator stehen.

39 XX XX

Analoger Befehl zu 3D XX XX. Der Unterschied liegt nur in der Verwendung des Y-Register's.

EOR

49 XX

Exklusives Oder

A	B	T
L	L	L
L	H	H
H	L	H
H	H	L

Es wird das Folgebyte mit dem momentanen Akkumulatorinhalt logisch EOR verknüpft. Das Ergebnis steht im Akku.

z.B. 49 31 momentaner Akkuinhalt sei z.B. 05
31 ≙ 0 0 0 1 1 1 1 1
05 ≙ 0 0 0 0 1 0 0 1
Nach der Befehlsausführung wird (0 0 0 1 0 1 1 0) 2 ≙ (16) Hex im Akku stehen.

4D XX XX

Es wird der Inhalt der Speicherzelle, die sich aus den 2 Folgebytes ergibt (Lower- und Higherbyte) mit dem momentanen Akkuinhalt log. EOR verknüpft. Das Ergebnis steht dann im Akku, wobei voriger Inhalt überschrieben wird.

z.B. 4D 02 03 Der Inhalt der Speicherzelle 0302 sei z.B. FE
Der momentane Akkuinhalt sei z.B. FF
Nach der Befehlsausführung wird im Akku 01 stehen.

45 XX

Der Inhalt der Speicherzelle, die durch das Folgebyte angegeben ist, wobei es sich um eine Adresse in der Zeropage handelt wird mit dem Akkumulator log. EOR verknüpft. Das Ergebnis wird im Akku geladen, wobei voriger Inhalt überschrieben würde.

z.B. 45 04 momentaner Akkuinhalt sei z.B. 03
Inhalt der Zelle 00 04 sei 02
Nach Befehlsausführung wird 06 im Akku stehen.

41 XX

Es wird der Inhalt des X-Register's zum Folgebytewert addiert. Das Additionsergebnis ist eine Adresse in der Zeropage. Deren Inhalt ist das Lowerbyte, wobei der nächstfolgende Adresseninhalt das Higherbyte ist. Der Inhalt aus der Adresse die sich aus Lower- und Higherbyte ergibt wird mit dem momentanen Akkumulatorinhalt log. EOR verknüpft. Das Ergebnis wird in den Akku geladen.

Anmerkung:

Ein eventuell vorkommender Überlauf bei der Addition X-Register und Folgebyte wird nicht berücksichtigt!

z.B. 41 32 X-Registerinhalt sei 01
momentanter Akkuinhalt sei z.B. 07
01 + 32 → 33
Inhalt der Adresse 0033 sei z.B. 01
Inhalt der Adresse 0034 sei z.B. 02
Der Inhalt der daraus resultierenden Adresse 02 01 sei z.B. 05
Nach der Befehlsausführung wird im Akku 02 stehen.

51 XX

Es wird der Inhalt der Adresse das das Folgebyte angibt zum Y-Register addiert. Das Additionsergebnis ist eine Adresse, deren Inhalt mit dem Akkuinhalt logisch EOR verknüpft wird. Das Ergebnis steht im Akku. Tritt aber bei der Addition ein Überlauf auf, so wird der Überlauf zu der nächstfolgenden Adresse die im Folgebyte angegeben ist, addiert. Dabei ist nun der Inhalt der Folgebyteadresse das Lowerbyte und der Inhalt der nachfolgenden Adresse und Überlauf das Higherbyte.
Der Inhalt der Adresse die sich aus Lower- und Higherbyte ergibt wird mit dem momentanen Akkuinhalt log. EOR verknüpft. Das Ergebnis steht im Akku.

ohne Überlauf:

z.B. 51 02 Inhalt der Speicherzelle 00 02 sei z.B. 01
Y-Registerinhalt sei z.B. 04; momentaner Akkuinhalt sei z.B. 08
01 + 04 → 05
Der Inhalt der Speicherzelle 05 sei z.B. 06
Nach der EOR Verknüpfung wird 0E im Akku stehen.

mit Überlauf:

z.B. 51 02 Inhalt der Speicherzelle 00 02 sei z.B. FF
Y-Registerinhalt sei z.B. 03
FF + 03 → 1 (Überlauf) 02 ist nun das Lowerbyte und die Addition des Zelleninhaltes Adresse nach der Folgebyteausgabe und Überlauf ist das Higherbyte; z.B. Inhalt der Adresse 03 ist z.B. 04 und Überlauf 1 dazuaddiert ergibt das Higherbyte 05. Lower- und Higherbyte ergeben die Adresse 05 02. Der Inhalt dieser Adresse wird mit dem Akkuinhalt log. EOR verknüpft.

Bemerkung:

Ist z.B. der Inhalt der Adresse 03 FF so ergibt deren Addition mit dem Überlauf wieder einen Überlauf. Doch dieser neuerlich erzeugte Überlauf wird nicht berücksichtigt.
Also FF + 1 → 1 00. Dann würde der Inhalt der Adresse 00 02 log. EOR verknüpft.

55 XX

Es wird das Folgebyte mit dem Inhalt des X-Register's addiert. Die Addition ist eine neue Adresse, deren Inhalt mit dem momentanen Akkuinhalt log. EOR verknüpft wird. Ergebnis steht im Akku.

z.B. 55 09 X-Registerinhalt sei z.B. 02
momentaner Akkuinhalt sei z.B. 01
09 + 02 → 0B
Inhalt der Adresse 0B sei z.B. 10
Nach der Befehlsausführung wird im Akku (11) Hex stehen.

5D XX XX

Es wird die Adresse, die sich aus den beiden Folgebytes ergibt (Lower- und Higherbyte) mit den Inhalt des X-Register's addiert.
Dieses Additionsergebnis ergibt eine neue Adresse, deren Inhalt mit dem momentanen Akkuinhalt log. EOR verknüpft wird.
Das Ergebnis steht im Akku.

z.B. 5D 00 01 X-Registerinhalt sei 01
01 00 + 01 → 01 01
Inhalt der Adresse 01 01 sei z.B. F0
momentaneer Akkuinhalt sei z.B. 0F
Nach der Befehlsausführung wird FF im Akkumulator geladen.

59 XX XX

Analoger Befehl zu 5D XX XX. Der Unterschied liegt in der Verwendung des Y-Register's.

ORA

09 XX

ODER FUNKTION

A	B	T
L	L	L
L	H	H
H	L	H
H	H	H

Es wird das Folgebyte mit dem momentanen Akkumulatorinhalt logisch Oder verknüpft. Ergebnis wir im Akku gespeichert.

z.B. 09 FE momentaner Akkuinhalt sei z.B. 01
Nach der Befehlsausführung wird FF der Inhalt des Akkumulator's sein.

0D XX XX

Es wird der Inhalt der Speicherzelle, die sich aus den beiden Folgebytes erbibt (Lower- und Higherbyte) mit dem momentanen Akkumulatorinhalt logisch ODER verknüpft.

z.B. 0D 00 01 momentaner Akkuinhalt sei z.B. 0A
Inhalt der Adresse 01 00 sei z.B. 0B
Nach der Befehlsausführung wird 0B der Akkumulatorinhalt sein.

05 XX

Es wird der Inhalt der Speicherzelle die sich aus dem Folgebyte ergibt mit dem momentanen Akkumulatorinhalt logisch ODER verknüpft. Das Ergebnis wird in Akku geladen.

z.B. 05 03 momentaner Akkuinhalt sei z.B. 08
Inhalt der Adresse 00 03 sei z.B. 09
Nach der Befehlsausführung wird der Akkuinhalt 09 sein.

01 XX

Es wird der Inhalt des X-Register's zum Folgebytewert addiert. Das Additionsergebnis ist eine Adresse in der Zeropage. Deren Inhalt ist

das Lowerbyte wobei der nächstfolgende Adresseninhalt das Higherbyte ist. Der Inhalt der aus Lower- und Higherbyte ergebenden Adresse wird mit dem Akkumulatorinhalt logisch ODER verknüpft. Ein eventueller Überlauf bei der Addition wird nicht berücksichtigt.

z.B. 01 05 X-Registerinhalt sei z.B. 08
05 + 08 → 0D
Inhalt der Adresse 000D sei z.B. 03
Inhalt der Adresse 000E sei z.B. 02
momentaner Akkuinhalt sei z.B 01
Inhalt der Adresse 02 03 sei z.B. 02
Nach der Befehlsausführung wird im Akkumulator 02 stehen.

11 XX

Es wird der Inhalt der Adresse des Folgebytes zum Y-Register addiert. Das Additionsergebnis ist eine Adresse deren Inhalt mit dem Akkumulatorinhalt logisch ODER verknüpft wird. Das Ergebnis steht anschließend im Akku.
Tritt bei der Addition ein Überlauf auf, d.h. Carry-Flag wird gesetzt, so wird dieser Überlauf zu der nächstfolgenden Adresse die im Folgebyte angegeben ist addiert.
Dabei ist der Wert ohne Überlauf das Lowerbyte und die Addition Überlauf und Inhalt der nachfolgenden Adresse das Higherbyte. Tritt hier ein Überlauf auf, so wird er nicht berücksichtigt. Der Inhalt der Adresse, die sich aus Lower- und Higherbyte ergibt, wird mit dem Akkumulatorinhalt logisch ODER verknüpft.
Das Ergebnis wird im Akku gespeichert.

ohne Überlauf:
z.B. 01 07 Y-Registerinhalt ist z.B. 04
Inhalt der Adresse 0007 ist z.B. 02
02 + 04 → 06
Der Inhalt der Speicherzelle 06 wird mit dem momentanen Akkumulatorinhalt log. ODER verknüpft. Ergebnis steht im Akkumulator.

mit Überlauf:
z.B. 01 07 Y-Registerinhalt ist z.B. 04
Inhalt der Adresse 0007 sei z.B. FC
FC + 04 → 1 (Überlauf) 01
Der Wert ohne Überlauf also 01 ist jetzt das Lowerbyte. Das Higherbyte ergibt sich aus der Addition

des Überlaufes mit dem Inhalt der nächstfolgenden Folgebyteadresse. Zu unserem Fall Inhalt der Adresse 0008 + Überlauf. Es soll z.B. Inhalt von Adresse 0008 03 sein. Das Higherbyte ist dann 03 + 1 → 04. Der Inhalt von der Lower- und Higherbyte ergebenden Adresse wird mit dem momentanen Inhalt des Akkumulator logisch ODER verknüpft und im Akku geladen.

Anmerkung:

Ist z.B. der Inhalt der Adresse 0008 = FF wird die Addition mit dem Überlauf wieder einen Überlauf erzeugen. Es wird aber der neuerliche Überlauf nicht berücksichtigt bzw. weg gelassen, so daß FF + 1 00 ergibt.

15 XX

Es wird das Folgebyte mit dem X-Registerinhalt addiert. Die Addition ergibt eine neue Adresse, deren Inhalt mit dem momenten Akkuinhalt logisch ODER verknüpft wird. Das Ergebnis der Verknüpfung steht dann im Akku.

z.B. 15 07 X-Registerinhalt sei z.B. 01
momentaner Akkumulatorinhalt sei z.B. 0F
07 + 01 → 08
Inhalt der Adresse 0008 sei z.B. FF
Nach der Befehlsausführung wird im Akkumulator FF stehen.

1D XX XX

Es wird die Adresse die sich aus Lower- und Higherbyte ergibt mit dem Inhalt des X-Register's addiert. Die Addition ergibt wiederum eine neue Adresse, wobei deren Inhalt mit dem momentanen Akkuinhalt log. ODER verknüpft wird. Das Ergebnis steht wiederum im Akkumulator.

z.B. 1D 00 17 X-Registerinhalt ist z.B. 01
momentaner Akkuinhalt sei z.B. 02
17 00 + 01 → 17 01
Inhalt von 17 01 sei z.B. FD
Nach der Ausführung des Befehls wird im Akkuinhalt FF stehen.

19 XX XX

Dieser Befehl entspricht dem 1D XX XX Befehl mit dem einen Unterschied, daß hier das Y-Register verwendet wird.

SBC

E9 XX

A - M - C → A

Es wird das Folgebyte vom momentanen Akkumulatorinhalt subtrahiert. Gleichzeitig wird von diesem Ergebnis noch der invertierte Wert des Carry-Flags subtrahiert.

Das Ergebnis wird im Akkumulator abgespeichert.

z.B. E9 02 momentaner Akkuinhalt sei FE

FE - 02 → FC

Nehmen wir an, daß das Carry-Flag vor der Befehlsausführung mit 0 gesetzt ist. Der invertierte Wert ergibt aber 1.

FC - 1 → FB

Der Akkumulatorinhalt nach der Befehlsausführung ist FB.

ED XX XX

Es wird der Inhalt der Adresse, die aus den beiden Folgebytes hervor geht vom momentanen Akkumulatorinhalt subtrahiert,gleichzeitigwird zu diesem Ergebnis der invertierte Wert des Carry-Flags subtrahiert.

z.B. ED 00 01 momentaner Akkuinhalt sei z.B. 0F

Inhalt der Adresse 01 00 soll 03 sein.

Carry-Flag 1 gesetzt d.h. 0F - 03 - 0 → 0C

wird im Akku abgespeichert.

E5 XX

Es wird der Inhalt der Adresse, die aus dem Folgebyte hervorgeht vom momentanen Akkumulatorinhalt subtrahiert. Der invertierte Carry-Flag-Wert wird anschließend vom Ergebnis noch subtrahiert.

z.B. E5 03 momentaner Akkuinhalt sei z.B. 04

Inhalt der Adtresse 0003 sei 01

Carry-Flag nicht gesetzt: d.h. 04 - 01 - 01 → 02

02 wird im Akku geladen.

E1 XX

Es wird der Inhalt des X-Register's zum Folgebyte addiert. Das Ergebnis dieser Addition ist eine Adresse in der Zeropage. Deren Inhalt ist das Lowerbyte und die nächstfolgende Adresse ist das Higherbyte. Der Inhalt der sich aus Lower- und Higherbyte ergebenden Adresse wird vom Akkumulatorinhalt subtrahiert.
Das invertierte Carry-Flag wird vom Ergebnis auch noch subtrahiert.

z.B. E1 03 X-Registerinhalt ist z.B. 04
03 + 04 → 07
Inhalt der Adresse 0007 ist z.B. BB
Inhalt der nächstfolgenden Adresse 0008 ist z.B. 01
Inhalt der somit errechneten Adresse 01 BB ist z.B. 01
momentaner Akkuinhalt sei z.B. 04
Carry-Flag gesetzt: 04 - 01 - 0 → 03
03 wird in Akku gespeichert.

F1 XX

Es wird der Inhalt der Adresse des Folgebytes zum Y-Registerinhalt addiert. Additionsergebnis ist eine neue Adresse, deren Inhalt mit dem investierten Carry-Flag vom momentanen Akkuinhalt subtrahiert wird. Tritt bei der Addition zu, Folgebyte und Y-Register ein Überlauf auf, so wird der Überlauf zu der nächstfolgenden Adresse die im Folgebyte angegeben ist addiert. Dabei ist der Wert ohne Überlauf das Lowerbyte und die Addition Überlauf und Inhalt der nächstfolgenden Adresse das Higherbyte. Tritt hier ein Überlauf auf, so wird er nicht berücksichtigt. Der Inhalt der Adresse die sich aus Lower- und Higherbyte erbibt und das invertierte Carry-Flag wird vom momentanen Akkuinhalt subtrahiert.

ohne Überlauf:

z.B. F1 07 Y-Registerinhalt sei z.B. 04
Inhalt der Speicherzelle 0007 sei z.B. 01
01 + 04 → 05
Der Inhalt der Speicherzelle 0005 sei z.B. 05
momentaner Akkuinhalt z.B. 0F
Carry-Flag nicht gesetzt.
Nach Befehlsausführung wird im Akku
0F - 05 - 01 → 09 stehen.
Carry-Flag ist gesetzt.

mit Überlauf:

z.B. F1 07 Y-Registerinhalt sei z.B. 04
Inhalt der Speicherzelle 0007 sei z.B. FE
FE + 04 → 1 (Überlauf) 02
02 ist Lowerbyte Folgebyte + 1 also 0008 sei z.B. 01 + Überlauf 1 ergibt 02; 02 ist Higherbyte
Lower- und Higherbyte ergibt Adresse 0202,
Inhalt z.B. 05; momentaner Akkuinhalt 0F
Carry-Flag gesetzt
Nach der Befehlsausführung wird 0F - 05 - 00 = 0A im Akku stehen. Carry-Flag ist gesetzt.

F5 XX

Es wird das Folgebyte mit dem Inhalt des X-Register's addiert. Die Addition ist eine neue Adresse, deren Inhalt mit dem invertierten Carry-Flag vom momentanen Akkuinhalt subtrahiert wird.

z.B. F5 07 X-Registerinhalt sei z.B. 01
07 + 01 → 08
Inhalt der Adresse 08 sei z.B. FE
momentaner Akkuinhalt sei z.B. 05
Carry-Flag nicht gesetzt.
Nach Befehlsausführung wird 05 - FE - 01 = also 06 (- 250) Dez im Akku stehen.
Carry-Flag nicht gesetzt.

FD XX XX

Es wird die Adresse, die sich aus den 2 Folgebytes ergibt (Lower- und Higherbyte) und der Inhalt des X-Register's addiert. Dieses Additionsergebnis ergibt eine neue Adresse, deren Inhalt mit dem invertierten Carry-Flag vom momentanen Akkuinhalt subtrahiert wird.

z.B. FD 00 01 X-Registerinhalt sei z.B. 02
0100 + 02 → 0102
Inhalt von 0102 sei z.B. 02; Akkuinhalt sei 0E,
Carry-Flag gesetzt.
Nach der Befehlsausführung wird im Akkuinhalt 0E - 02 - 0 → 0C stehen. Carry-Flag bleibt gesetzt.

F9 XX XX

Dieser Befehl ist der analoge Befehl zum FD XX XX Befehl, nur daß hier das Y-Register angesprochen wird.

DEC

CE XX XX

Es wird der Inhalt der Speicherzelle, die sich aus den 2 Folgebytes ergibt (Lower- und Higherbyte) um 1 erniedrigt und in derselben Adresse wieder abgelegt.

z.B. CE 00 01 der Inhalt von Adresse 0100 soll z.B. 05 sein
Von diesen 5 wird 1 subtrahiert und in Speicherzelle 0100 gespeichert.

C6 XX

Es wird der Inhalt der Speicherzelle des Folgebytes (Zeropage) um 1 erniedrigt und in derselben Adresse wieder abgelegt.

z.B. C6 01 Der Inhalt der Adresse 0001 wird um 1 erniedrigt und in derselben Speicherzelle abgelegt
also Inhalt von 0001 vorher sei 07
Inhalt von 0001 nachher ist 06

D6 XX

Es wird das Folgebyte mit dem Inhalt des X-Register's addiert. Die Addition ergibt eine neue Adresse, deren Inhalt um 1 erniedrigt wird. Tritt ein Überlauf bei der Addition auf so wird der Überlauf nicht berücksichtigt.

z.B. D6 08 X-Registerinhalt sei 01
08 + 01 → 09
Inhalt der Adresse 09 sei z.B. FF
dieser Inhalt wird um 1 erniedrigt, so daß danach in Speicherzelle 0009 der Inhalt FE steht.

DE XX XX

Es wird die Adresse, die sich aus den 2 Folgebytes ergibt (Lower- und Higherbyte), und der Inhalt des X-Register's addiert. Die Addition ergibt eine neue Adresse deren Inhalt um 1 erniedrigt wird.

z.B. DE 00 11 X-Registerinhalt sei 01
1100 + 01 → 1101
Inhalt der Adresse 1101 z.B. 08 wird um 1 erniedrigt so daß danach 07 darin steht.

DEX

CA
Es wird der Inhalt des X-Register's um 1 erniedrigt und in X-Register wieder abgespeichert.

z.B. X-Registerinhalt 05
Nach Ausführung des Befehls steht 04 im X-Register.



88
Es ist der analoge Befehl zum DEX-Befehl nur daß hier das Y-Register verwendet wird.

INC

EE XX XX

Es wird der Inhalt der Speicherzelle die sich aus den 2 Folgebytes ergibt (Lower- und Higherbyte) um 1 erhöht und wieder in der selben Adresse abgelegt.

z.B. EE 00 02 Inhalt der Zelle 0200 soll 08 sein. Nach der Ausführung steht 09 in der Zelle 0200.

E6 XX

Es wird der Inhalt der Speicherzelle die das Folgebyte angibt um 1 erhöht und in derselben Zelle abgelegt.

z.B. E6 D0 Inhalt von 00D0 sei 08
Nach der Ausführung steht in Adresse 00D0 09.

F6 XX

Es wird das Folgebyte mit dem Inhalt des X-Register's addiert. Das Ergebnis ist eine Adresse in der Zeropage. Deren Inhalt wird um 1 erhöht und in der gleichen Adresse abgelegt.
Ein eventuell vorkommender Überlauf bei der Addition wird nicht berücksichtigt.

z.B. F6 06 X-Registerinhalt sei 02
06 + 02 → 08
Inhalt der Speicherzelle 0008 sei 01.
Nach Ausführung des Befehls steht 0B in Adresse 0008.

FE XX XX

Es wird die Adresse die sich aus den beiden Folgebytes ergibt (Lower- und Higherbyte) und der Inhalt des X-Register's addiert. Das Ergebnis der Addition ist eine Adresse, deren Inhalt um 1 erhöht wird. Das Ergebnis steht wieder in derselben Speicherzelle.

z.B. FE 01 03 X-Registerinhalt sei 05
0301 + 05 → 0306
der Inhalt von 0306 soll z.B. 0F sein. Nach Ausführung des Befehls steht in Speicherzelle 0301 10.



E8

Es wird der Inhalt des X-Register's um 1 erhöht und anschließend im X-Register wieder gespeichert.

z.B. momentaner X-Registerinhalt 09
Nach Ausführung des Befehls steht 0A im X-Register.



C8

Der analoge Befehl zum INX-Befehl. Der Unterschied liegt nur in der Benutzung des Y-Register's.

ASL

OE XX XX

C ← [7 0] ← 0

Es wird der Inhalt der Speicherzelle die sich aus den Folgebytes ergibt (Lower- und Higherbyte) um 1 Stelle nach links verschoben wobei von der rechten Seite eine 0 eingeschoben wird und das herausgeschobene Bit im Carry-Flag gespeichert wird.

z.B. 0E 00 03 Inhalt von 0300 sei z.B. (81) 16 = (1000 0001) nach der Ausführung des Befehls wird in der Speicherzelle 0300 die Zahl 2 stehen und das Carry-Flag ist mit 1 gesetzt.

Bildlich: Zelle 300

| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | vorher

Carry-Flag ← | 1 | | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | nachher
ist 1

06 XX

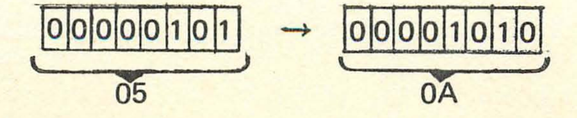
Es wird der Inhalt der Speicherzelle die durch das Folgebyte angegeben ist um 1 Stelle nach links verschoben unter gleichzeitiger Aufnahme von der rechten Seite einer 0.

z.B. 06 04 Inhalt von 04 sei 1
nach der Ausführung steht in Adresse 0004 eine 2 und Carry-Flag ist 0.
Siehe auch 0E XX XX

0A

Der Akkumulatorinhalt wird unter gleichzeitiger Einschiebung einer 0 von rechts nach links verschoben. Das Carry-Flag wird mit dem herausfahlenden Bit gesetzt.

z.B. Akkuinhalt sei 05
Nach der Befehlsausführung steht 0A im Akku.



16 XX

Es wird das Folgebyte mit dem Inhalt des X-Register's addiert. Das Ergebnis ist eine neue Adresse deren Inhalt unter Einschiebung von rechts einer 0 um 1 Stelle nach links verschoben wird das herausfallende Bit wird im Carry-Flag gesetzt.

z.B. 16 05 X-Registerinhalt sei 08
05 + 08 → 0D
angenommener Inhalt von 000B sei 80
Nach der Ausführung ist in 000D die Zahl 0 gespeichert. Das Carry-Flag ist mit 1 gesetzt worden.

1	0	0	0	0	0	0	0

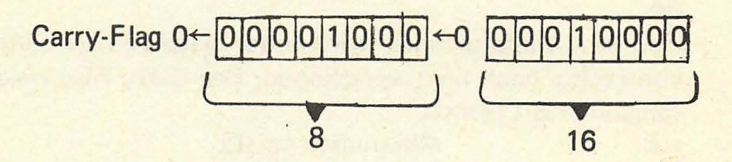
0	0	0	0	0	0	0	0

Ein eventueller Überlauf bei der Addition wird nicht berücksichtigt!

1E XX XX

Es wird die Adresse die sich aus den 2 Folgebytes ergibt (Lower- und Higherbyte) und der Inhalt des X-Register's addiert. Das Additionsergebnis ist eine Adresse, deren Inhalt nach links verschoben wird. Rechts wird eine 0 eingeschoben und der linke Überlauf wird im Carry-Flag gesetzt.

z.B. 1E 00 01 X-Registerinhalt sei z.B. 05
0100 + 05 → 0105
Der Inhalt der Adresse 0105 sei z.B. 08
Nach der Befehlsausführung steht in der Adresse 105 die 10



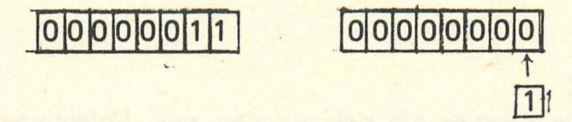
LSR

4E XX XX

0 → [7 0] → C

Es wird der Inhalt der Speicherzelle die sich aus den Folgebytes erbibt (Lower- und Higherbyte) um 1 Stelle nach rechts verschoben, wobei von der linken Seite eine 0 eingeschoben wird und der Überlauf der Verschiebung im Carry-Flag gesetzt ist.

z.B. 4E 00 02 Inhalt von 0200 sei z.B. (03) 16 = (0000 0011)
nach der Ausführung des Befehls wird in Adresse 0200 eine 1 der Inhalt sein. die 1 die hinausgeschoben würde steht nun im Carry-Flag.



46 XX

Es wird der Inhalt der Speicherzelle die durch das Folgebyte angegeben ist um 1 nach rechts verschoben unter gleichzeitiger Aufnahme eine 0 von der linken Seite.

z.B. 06 01 Inhalt momentan sei die Hex Zahl 0F
nach der Ausführung des Befehls wird 07 der Inhalt von Adresse 0001 sein und das Carry-Flag wird 1 gesetzt sein.

4A

Der Akkumulatorinhalt wird unter gleichzeitiger Einschiebung einer 0 von links nach rechts verschoben. Das somit frei werdende Bit von der rechten Seite wird im Carry-Flag gesetzt.

z.B. Inhalt des Akku sei z.B. FE
Nach der Ausführung des Befehls wird im Akku 7F stehen und das Carry-Flag ist mit 0 gesetzt.

[1 1 1 1 1 1 1 0] [0 1 1 1 1 1 1 1] → [0] Carry-Flag

56 XX

Es wird das Folgebyte mit dem Inhalt des X-Register's addiert. Das Additionsergebnis ist eine neue Adresse, deren Inhalt um 1 nach rechts verschoben wird unter gleichzeitiger Aufnahme eine 0 von der linken Seite. Das Carry-Flag wird mit dem freiwerdenden Bit von der rechten Seite gesetzt.

Bemerkung:

Ein eventueller Überlauf bei der Addition wird nicht berücksichtigt.

z.B. 56 FE X-Registerinhalt 0F

FE + 0F → 10D

Da der Überlauf nicht berücksichtigt wird, wird der Inhalt der Adresse 000D um 1 Stelle nach rechts verschoben.

5E XX XX

Es wird die Adresse die sich aus den zwei Folgebytes ergibt (Lower- und Higherbyte) und der Inhalt des X-Register's addiert. Das Additionsergebnis ist eine Adresse, deren Inhalt nach rechts um eine Stelle verschoben wird. Das rechte Bit das dabei frei wird, wird im Carry-Flag gesetzt.

z.B. 5E 00 01 X-Registerinhalt sei z.B. 08

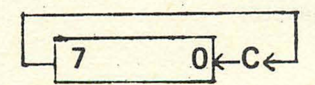
Der Inhalt der Adresse 0108 sei z.B. 07

Nach der Befehlsausführung steht in der Adresse 108 der Inhalt 03; das Carry-Flag ist mit 1 gesetzt.

0 → | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | → | 1 | Carry-Flag

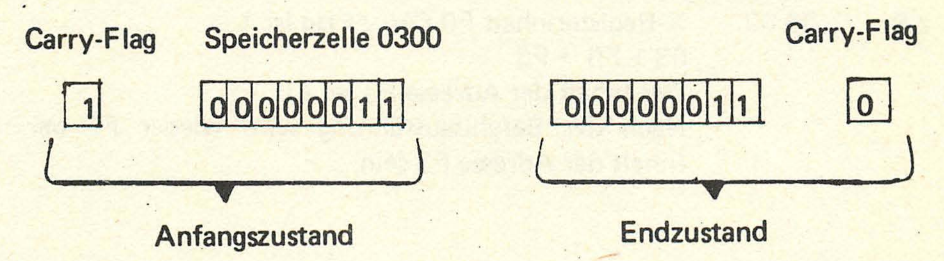
ROL

2E XX XX



Es wird der Inhalt der Speicherzelle, die sich aus den Folgebytes ergibt (Lower- und Higherbyte) rolliert, d.h. die linke Bit wird in das Carry-Flag gebracht, wobei der vorige Carry-Flag-Inhalt rechts wieder eingeschoben wird.

z.B. 2E 00 03 Inhalt von 0300 sei 01,
das Carry-Flag sei 1.
Nach der Ausführung des Befehls wird in der Speicherzelle 0300 der Inhalt 03 sein. Das Carry-Flag wird anschließend 0 gesetzt sein.



26 XX

Es wird der Inhalt der Speicherzelle die durch das Folgebyte angegeben ist folgender Änderungen unterzogen, unter zu Hilfe nahme des Carry-Flaggs wird Speicherzelleninhalt nach links verschoben in das Carry-Flag hinein, der vorige Carry-Flag-Inhalt wird von rechts in den Speicherzelleninhalt geschoben.

z.B. 26 0F Inhalt von 0F sei z.B. 02
Das Carry-Flag sei z.B. 0.
Nach der Ausführung wird der Inhalt von 0F 04 sein. Das Carry-Flag wird wieder mit 0 geladen.

2A

Der Akkumulatorinhalt wird um eine Stelle nach links verschoben. Der momentane Inhalt des Carry-Flags wird von rechts eingeschoben; das Bit das hinausgeschoben wurde wird im Carry-Flag neu gesetzt.

z.B. Akkuinhalt FF, Carry-Flaginhalt sei 0
Nach der Befehlsausführung steht FE als Inhalt im Akku. Das Carry-Flag ist 1 gesetzt.

36 XX

Es wird das Folgebyte mit dem Inhalt des X-Registers addiert. Das Ergebnis ist eine neue Adresse, die um eine Stelle nach links verschoben wird. Der momentane Carry-Flag-Inhalt wird von rechts eingeschoben. Das Bit das aus der Speicherzelle hinausgeschoben wurde wird als neues Bit im Carry-Flag gesetzt.

Bemerkung:
Ensteht bei der Addition ein Überlauf, so wird er nicht berücksichtigt.

z.B. 36 03 X-Registerinhalt F0 Carry-Flag ist 1
03 + F0 → F3
Der Inhalt der Adresse F3 sei z.B. FF.
Nach der Befehlsausführung wird wieder FF der Inhalt der Adresse F3 sein.

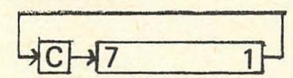
3E XX XX

Es wir die Adresse, die sich aus den 2 Folgebytes ergibt (Lower- und Higherbyte) und der Inhalt des X-Register's addiert. Das Additionsergebnis ist eine neue Adresse, deren Inahlt um 1 Stelle nach links verschoben wird. Der momentane Inhalt des Carry-Flags wird von rechts eingeschoben. Das Bit, das aus der Speicherzelle hinausgeschoben wurde, wird im Carry-Flag neu gesetzt.

z.B. 3E 00 01 X-Registerinhalt sei z.B. 05
0100 + 05 → 0105
Der Inhalt der Adresse 0105 sei z.B. 04, Carry-Flag sei 0. Nach der Befehlsausführung steht in der Adresse 105 08 und das Carry-Flag wird wieder 0 gesetzt.

ROR

6E XX XX



Es wird der Inhalt der Speicherzelle, die sich aus den Folgebytes ergibt (Lower- und Higherbyte) um eine Stelle nach rechts verschoben. Der momentane Inhalt des Carry-Flags wird von links eingeschoben und das Bit das hinausgeschoben wurde ist der neue Inhalt des Carry-Flags.

z.B. 6E 00 01 Inhalt der Speicherzelle 100 sei z.B. 02, Carry-Flag-Inhalt sei 1. Nach der Befehlsausführung steht 81 im Inhalt der Speicherzelle 100. Das Carry-Flag ist anschließend mit 0 gesetzt.

66 XX

Es wird der Inhalt der durch das Folgebyte angegebenen Speicherzelle in der Zeropage um 1 Stelle nach rechts geschoben. Der momentane Inhalt des Carry-Flags wird von links eingeschoben und das Bit, das aus der Speicherzelle rechts hinausgeschoben wurde ist der neue Inhalt des Carry-Flags.

z.B. 66 E0 Inhalt von E0 sei z.B. 04, Carry-Flag sei z.B. 0 Nach der Ausführung des Befehls wird der Inhalt von Speicherzelle E0 02 sein. Das Carry-Flag wird nach wie vor 0 sein.

6A

Der Akkumulatorinhalt wird um eine Stelle nach rechts verschoben. Der momentane Inhalt des Carry-Flags wird von links eingeschoben. Das Bit das hinausgeschoben wurde, ist der neue Inhalt des Carry-Flags.

76 XX

Es wird das Folgebyte mit dem Inhalt des X-Register's addiert. Entsteht bei dieser Addition ein Überlauf, so wird er nicht berücksichtigt. Das Additionsergebnis wird um 1 Stelle nach rechts verschoben. Der momentane Carry-Flag-Inhalt wird von links in die Speicherzelle eingeschoben. Das Bit das aus der Speicherzelle hinausgeschoben wurde ist neuer Inhalt des Carry-Flags.

z.B. 76 0F X-Registerinhalt sei 01, Carry-Flag soll 1 gesetzt sein.
0F + 01 → 10
Der Inhalt von 0010 soll z.B. FE sein.
Nach der Befehlsausführung wird FF in der Speicherzelle stehen und 0 wird im Carry-Flag sein.

7E XX XX

Es wird die Adresse, die sich aus den zwei Folgebytes ergibt (Lower- und HigherByte) un der Inhalt des X-Register's addiert. Das Additionsergebnis wird um 1 Stelle nach rechts verschoben. Der momentane Carry-Flag-Inhalt wird von links in die Speicherzelle eingeschoben. Das Bit das bei der Verschiebung der Speicherzelle hinausgeschoben wurde, ist nun neuer Inhalt des Carry-Flags.

z.B. 7E 00 02 X-Registerinhalt sei z.B. 01
0200 + 01 → 0201
Der Inhalt der Speicherzelle 0201 sei z.B. 10. Das Carry-Flag soll 0 gesetzt sein. Der Inhalt von 0201 ist nach der Befehlsausführung 08. Das Carry-Flag bleibt auf 0 gesetzt.

CMP

A - M

Allgemeines:

Bei diesem Befehl wird nur das Statusregister geändert. Alle anderen Registerinhalte bleiben erhalten. Der CMP-Befehl wird nur im Zusammenhang mit Verzweigungsbefehlen werwendet.

BCC ermittelt A ⟨ M
BEQ ermittelt A = M
BEQ folgend auf BCS A ⟩ M
BCS folgend auf BCS A ⟩ M

C9 XX

Es wird das Folgebyte vom momentanen Akkumulatorinhalt subtrahiert. Nach dem erläuterten Schema werden die 3 Flags des Statusregister's gesetzt.

CD XX XX

Es wir der Inhalt der Speicherzelle, die sich aus den Folgebytes ergibt vom momentanen Akkuinhalt subtrahiert. Nach dem erläuterten Schema werden die 3 Flags des Statusregister's gesetzt.

C5 XX

Es wird der Inhalt der Speicherzelle die im Folgebyte angegeben ist vom momentanen Akkuinhalt subtrahiert und das Ergebnis wird nach unten erläuterten Schema benutzt, die 3 Statusregisterflags zu setzen.

Schema der Flagsetzung

	N	C	Z
Akku ⟨ Mem.	Bit 7=1→Set Bit 7=0→Reset	Reset	Reset
Akku = Mem.	Bit 7=1→Set Bit 7=0→Reset	Set	Set
Akku ⟩ Mem.	Bit 7=1→Set Bit 7=0→Reset	Set	Reset

C1 XX

Es wird der Inhalt des X-Registers's zum Folgebyte addiert. Das Ergebnis dieser Addition ist eine Adresse in der Zeropage. Deren Inhalt ist das Lowerbyte und die nächstfolgende Adresse ist das Higherbyte. Der Inhalt der Adresse die sich aus Lower- und Higherbyte ergibt wird vom momentanen Akkumulatorinhalt subtrahiert. Das Ergebnis setzt nach der Tabelle die 3 Flags des Statusregisters.

D1 XX

Es wird der Inhalt der Adresse, die das Folgebyte ergibt zumY-Registerinhalt addiert. Dies ergibt eine neue Adresse, deren Inhalt vom Akkumulator subtrahiert wird.
Tritt aber bei der Addition zu Folgebyteadresseninhalt und Y-Register ein Überlauf auf, so wird der Überlauf zu der nächstfolgenden Adresse die im Folgebyte angegeben ist addiert. Dabei ist der Wert ohne Überlauf das Lowerbyte und die Addition Überlauf und Inhalt der nächstfolgenden Adresse das Higherbyte. Tritt hier ein Überlauf auf so wird er nicht berücksichtigt. Der Inhalt der Adresse die sich aus Lower- und Higherbyte ergibt wird vom Akkumulatorinhalt subtrahiert. Das Ergebnis legt die Flagsetzung des Statusregister's fest.

D5 XX

Es wird das Folgebyte mit dem Inhalt des X-Register's addiert. Die Addition ist eine neue Adresse, deren Inhalt vom momentanen Akkuinhalt subtrahiert wird. Das Ergebnis setzt die Zustände der 3 Flags im Statusregister fest.

DD XX XX

Es wird die Adresse, die sich aus den 2 Folgebyter ergibt, Lower- und Higherbyte mit dem X-Registerinhalt addiert. Das Additionsergebnis ergibt eine neue Adresse, deren Inhalt vom momentanen Akkuinhalt subtrahiert wird. Dieses Ergebnis setzt die Zustände der 3 Flags des Statusregister's fest.

D9 XX XX;

Ist der analoge Befehl zu DD XX XX, nur im Unterschied dazu, daß das Y-Register verwendet wird.

CPX

X - M

Allgemeines:

Bei diesen Befehl wird kein Register oder Speicherinhalt geändert. Der CPM-Befehl wird nur im Zusammenhang von Verzweigungen verwendet.

BCC ermittelt A < M
BEQ ermittelt A = M
BEQ folgend auf BCS A > M
BCS folgend auf BCS A > M

E0 XX

Es wird das Folgebyte vom X-Registerinhalt subtrahiert. Nach dem erläuterten Schema werden die 3 Flags des Statusregisters gesetzt.

EC XX XX

Es wird der Inhalt der Speicherzelle, die sich aus den Folgebytes ergibt, vom X-Registerinhalt subtrahiert. Nach dem erläuterten Schema werden die 3 Flags des Statusregister's gesetzt.

E4 XX

Es wird der Inhalt der Speicherzelle die sich aus dem Folgebyte ergibt, vom momentanen X-Registerinhalt subtrahiert. Nach dem unten erläuterten Schema werden die Flags gesetzt.

Schema der Flagsetzung

	N	C	Z
X-Register < Mem.	Bit 7=1 → Set Bit 7=0 → Reset	Reset	Reset
X-Register = Mem.	Bit 7=1 → Set Bit 7=0 → Reset	Set	Set
X-Register > Mem.	Bit 7=1 → Set Bit 7=0 → Reset	Set	Reset

z.B. X-Register 04
E0 05 → Statusregister (60) Hex = (176) Dez
|1|0 1 1 0 0|0|0|

X-Register 05
E0 05 → Statusregister (33) Hex = (51) Dez
0 0 1 1 0 0|1|1
x| x

X-Register 06
E0 05 → Statusregister (31) Hex = (49) Dez
|0|0 1 1 0 0|0|1

CPY

Allgemeines:

Bei diesem Befehl wird kein Adresseninhalt außer dem Statusregister geändert. Der CPM-Befehl wird nur im Zusammenhang mit Verzweigungsbefehlen verwendet.

BCC ermittelt A i M
BEQ ermittelt A = M
BEQ folgend auf BCS A ⟩ M
BCS folgend auf BCS A ⟩ M

C0 XX

Es wird das Folgebyte vom Y-Register subtrahiert. Nach dem erläuterten Schema werden die 3 Flags des Statusregisters gesetzt.

CC XX XX

Es wird der Inhalt der Speicherzelle, die sich aus den Folgebytes ergibt, vom Y-Register subtrahiert. Nach dem erläuterten Schema werden die 3 Flags des Statusregister's gesetzt.

C4 XX

Es wird der Inhalt der Speicherzelle, die sich aus den Folgebyte ergibt vom Y-Registerinhaltsubtrahiert. Das Ergebnis wird zum setzen der 3 Flags benutzt.

Schema der Flagsetzung

	N	C	Z
Y-Register ⟨ Mem.	Bit 7=1 Set Bit 7=0 Reset	Reset Set	Reset Set
X-Register = Mem.	Bit 7=1 Set Bit 7=0 Reset	Set	Set
X-Register ⟩ Mem.	Bit 7=1 Set Bit 7=0 Reset	Set	Reset

z.B. Y-Register 04
C0 05 → Statusregister (60) Hex
1 0 1 1 0 0 0 0

Y-Register 05
C0 05 → Statusregister (33) Hex
0 0 1 1 0 0 1 1

Y-Register 06
C0 05 → Statusregister (31) Hex
0 0 1 1 0 0 0 1



2C XX XX

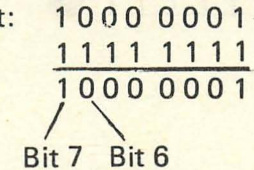
UND FUNKTION

A	B	Y
L	L	L
L	H	L
H	L	L
H	H	H

Der Inhalt der Speicherzelle die sich aus den beiden Folgebytes ergibt (Lower- und Higherbyte) wird mit dem Inhalt des Akkumulator's logisch UND verknüpft. Bit 6 und Bit 7 der Verknüpfung werden im Prozessorstatusregister gesetzt.

z.B. 2C 00 01 angenommen der Inhalt der Speicherzelle 0100 sei 81. Der angenommene Inhalt des Akkus sei FF.

UND verknüpft:



Das Prozessorstatusregister zeigt nach der Operation folgenden Inhalt

10XX XXXX /X bedeutet beliebig

bei unseren Test B0 ≙ 1 0 1 1 0 0 0 0

24 XX

Der Inhalt der Speicherzelle die durch das Folgebyte angegeben ist wird mit Akkumulatorinhalt logisch UND verknüpft. Bit 6 und Bit 7 der Verknüpfung werden im Statusregister abgelegt.

z.B. 24 03 angenommener Inhalt von 0003 sei C1 und Akkuinhalt sei FF.

UND verknüpft: 1 1 0 0 0 0 0 1 ≙ C1
1 1 1.1 1 1 1 1 ≙ FF
1 1 0 0 0 0 0 1

Bit 7 Bit 6

Statusregister hat nach der Operation folgenden Inhalt:

11XX XXXX bei unseremTest also 1111 0000

BCC

90 XX

Es wird das Carry-Flag Überlauf 6 + 7 auf 0 überprüft. Wenn die Bedingung erfüllt ist wird zu der Adresse gesprungen die sich aus dem Folgebyte. ergibt. Es können (127) 10 Adressen vorwärts und (128) 10 Adressen rückwärts gesprungen werden.
Bemerkung:

Das Carry-Flag befindet sich (KIM-1) im Statusregister und ist das Bit 0.

BCS

B0 XX

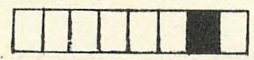
Es wird das Carry-Flag auf 1 überprüft. Ansonsten gilt das gleiche wie bei BCC.

BEQ

F0 XX

Es wird das Zero-Flag auf 1 überprüft. Tritt die Bedingung zu, so wird zu der Adresse gesprungen die sich aus dem Folgebyte ergibt (127) 10 Vorwärtssprünge und (128) 10 Rückwärtssprünge sind möglich.
Bemerkung:

Das Zero-Flag befindet sich (KIM-1) auch im Statusregister und ist Bit 1



z.B. F0 F4 falls das Zero-Flag 1 gesetzt ist, wird 4 Adressen rückwärts gesprungen.

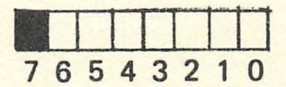
BNE

D0 XX
Es wird das Zero-Flag auf 0 überprüft. Ansonsten der selbe Verlauf wie beim BEQ-Befehl.

BMI

30 XX
Es wird das Negativ-Flag auf 1 überprüft. Trifft die Bedingung zu so wird zu der Adresse gesprungen die sich aus den Folgebyte ergibt.
Bemerkung:

Das Negativ-Flag befindet sich auch im Statusregister und ist dort beim KIM-1 das 7. Bit.



Statusregister

BPL

10 XX
Es wird das Negativ-Flag auf 0 überprüft. Ansonsten der selbe Befehl wie BMI.

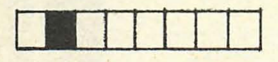
BVC

50 XX

Es wird das Overflow-Flag auf 0 überprüft. Trifft die Bedingung zu, so wird zu der Adresse gesprungen die sich aus dem Folgebyte ergibt.

Bemerkung:

Das Overflow-Flag befindet sich auch im Statusregister und ist dort beim KIM-1 das 6. Bit.



7 6 5 4 3 2 1 0

BVS

70 XX

Es wird das Overflow-Flag auf 1 überprüft. Ansonsten der selbe Befehl wie BVC.

Beispiel zu den Verzweigungsbefehlen:

Berechnung der relativen Adresse

Es soll am BNE-Befehl erklärt werten. Trifft sonst für alle Ver. Befehle zu.

```
                                    Ziel
                                    (Sprung)
Rückwärtssprung
     0200  E1       LDX FF          AZ  FF
     0202           DEX             CA
     0203           BNE  E1         DO  FD
     0205           END             00


Vorwärtssprung
     0300           LDX  03         A2  01
     0302  E2       DEX             CA
     0303           BEQ  E3         F0  03
     0305           JMP  E2         4C  02 02
     0308  E3       END             00
                                    (Ziel)
                                    Sprung
```

Beide Programme machen das gleiche; sie zählen jeweils das X-Register das Anfangs mit FF geladen wurde auf 0 herunter.

Zu der Sprungadressenbestimmung in den Verzweigungsbefehlen.

Vorwärtssprung: Es werden die zu überspringenden Adressen ab dem Folgebyte in der Verzweigeanweisung. In unserem Fall sind es 3 Adressen die übersprungen werden. Also 03 ist das Folgebyte.

Rückwärtssprung: Wie beim Vorwärtssprung wird die Anzahl der zu überspringenden Adressen vom Folgebyte des Verzweigungsbefehles aus gezählt. Da es sich aber um einen Rückwärtssprung handelt muß man folgendermaßen vorgehen. Ziehen Sie die ermittelten Adresssprünge , in unserem Beispiel 3 von der Binärzahl 1111 1111 ab und addieren sie zu den Ergebnis eine 1 Dazu.

Vorsicht mit den Zahlensystemen!

In unserem Beispiel:

```
 1111 1111
-0000 0011
----------
 1111 1100
+        1
----------
 1111 1101 ≙ FD
```

FD ist das Folgebyte

BRK

00

Das Break-Flag wird automatisch im Statusregister gesetzt.
Es wird unterschieden zwischen einem Programmbreak und einem Hardware-Interrupt. Keine anderen Benutzerinstruktionen werden modificiert.



4C XX XX

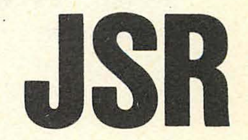
Absoluter Sprung zu der Adresse, die im Operanden in Lower- und Higherbyte angegeben ist.

z.B. 4C 00 02 Es wird bei Addresse 0200 im Programm fortgefahren

6C XX XX

Indirekter Sprung zu der Adresse, deren Lower- und Higherbyte sich folgendermaßen ergibt. Die 2 Folgebytes sind 1 Adresseangabe, deren Inhalt das Lowerbyte ist. Der nächstfolgende Adresseninhalt ist das Higherbyte.

z.B. 6C 01 03 Inhalt der Adresse 0301 sei z.B. 00
Inhalt der Adresse 0302 sei z.B. 02
somit erfolgt Sprung an Adresse 0200



20 XX XX

Es erfolgt ein absoluter Sprung in eine unterroutine, die entweder selbst eingegeben wurde oder bereits fest besteht. Die Folgebytes sind Lower- und Higherbyte der Sprungadresse.

z.B. 20 19 1F Es erfolgt ein Unterroutinensprung zu Start - Adresse 1F 19.

RTS

60

Bei selbst geschriebenen Unterroutinen muß der letzte Befehl ein RTS-Befehl sein. Das bedeutet daß im eingentlichen Programm fortgefahren wird.

RTI

40

Zurückspeichern des Statusregister und des Programmcounter welche beide im Stack gespeichert waren. Neufestlegung des Stackpointer Eigentliche Bedeutung Rücksprung von einem Interrupt.

NOP

EA

Es wird keine Operation ausgeführt. Alle Registerinhalte bleiben erhalten. Es wird zur Zeitverzögerung oder zum Programmaufüllen verwendet.

CLC

18

Das Carry-Flag wird 0 gesetzt.

SEC

38
Das Carry-Flag wird 1 gesetzt.

CLD

D8
Das Dezimal-Mode-Flag wird 0 gesetzt. Es wird in Hex gerechnet.

SED

F8
Das Dezimal-Mode-Flag wird 1 gesetzt. Es wird in Dezimal gerechnet.

CLI

58
Das IRQ-Disable-Flag wird 0 gesetzt.

SEI

78
Das IRQ-Disable-Flag wird 1 gesetzt.

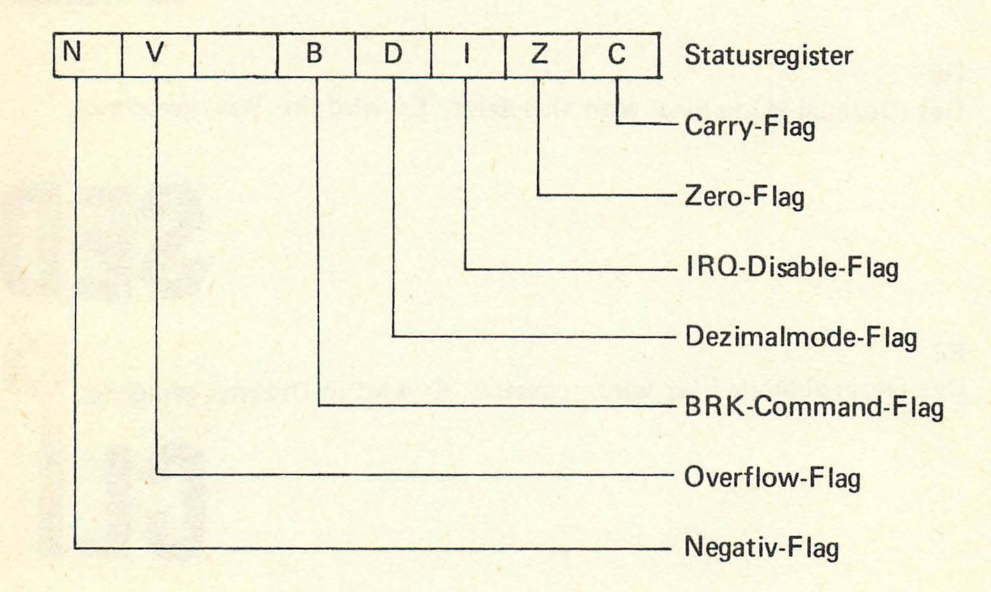


B8

Das Overflow-Flag wird 0 gesetzt.

Die angesprochenen Flags sind beim KIM-1 alle im Statusregister

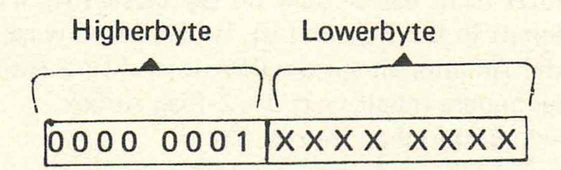
Folgende Zuordnung gilt:



PHA

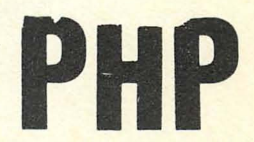
48

Der Inhalt des Akkumulator's wird in die von Stackpointer angegebene Adresse gespeichert. Der Stackpointer wird um 1 heruntergezählt.
Der Stackpointer hat folgendes Format



Die 1 ist vorgegeben.

z.B. Akkuinhalt sei z.B. BB
Stackpointerinhalt sei z.B. FC
Akkuinhalt BB wird also in Adresse 01 FC gespeichert. Anschließend ist Stackponterinhalt FB.



08

Der Inhalt des Prozessorstatusregister's wird in die vom Stackpointer angegebene Adresse gespeichert. Der Stackpointer wird um 1 heruntergezählt.
Der Stackpointer hat folgendes Format

0000 0001	XXXX XXXX

Die 1 ist Vorgegeben.

z.B. Nach einer Poeration sei der Statusregisterinhalt B0
Stackponterinhalt sei z.B. AA
d.h. Statusregisterinhalt B0 wird in Adresse 01AA gespeichert. Nach der Befehlsausführung wird im Stackpointer A9 stehen.

PLA

68

Diese Instruktion addiert 1 zum Inhalt des Stackpointer und gebraucht diesen Inhalt als Adresse. Der Inhalt dieser Adresse wird in Akku geladen.

Diese Instruktion benutzt nicht das C- oder 0-Flag. DasN-Flag wird gesetzt wenn der Akkuinhalt in Bit 7 gesetzt ist. Ist Bit 7 0 so wird es zurückgesetzt auf 0. ist der Akkuinhalt aus der PLA-Instruktion 0 so wird dasZ-Flag gesetzt. Jeder andere Inhalt setzt das Z-Flag zurück.

z.B. Stackpointerinhalt sei z.B. 03
Es wird 1 dazuaddiert, ergibt 04
Der Inhalt der Adresse 0104 sei z.B. FF somit wird FF im Akku geladen, das N-Flag ist auf 1 gesetzt.

Erklärung zu Adresse 0104

Die Adresse 0104 setzt sich aus dem Stackpointer zusammen. Da der Stackpointer ein Doppelwort ist wobei 1. Wort mit 0000 0001 festgelegt ist kommt diese Adresse zustande.

PLP

Z8

Diese Instruktion addiert 1 zum Inhalt des Stackpointer's und gebraucht dessen Inhalt als Adresse. Der Inhalt dieser Adresse wird in Prozessorstatusregister geladen. Diese Instruktion beeinflußt also alle Flags im Statusregister.

z.B. Stakcpointerinhalt sei z.B. 04
Es wird 1 dazuaddiert; ergibt 05
Der Inhalt der Adresse 0105 sei z.B. FE
Somit wird FE in Statusregister geladen. Alle Flags außer dem C-Flag sind gesetzt.

Randbemerkung:

Wenn der Inhalt der Adresse 105 z.B. 00 wäre so wäre der Inhalt des Statusregister's nur während dem Programmablauf 0. Sie können die 0 aber nicht einsetzen, da Sie dazu ein Break-Befehl eingeben müßten. Ein Breack-Befehl einerseits aber setzt das BRK-Flag auf 1 , womit der Wert der Anzeige verändert wäre.

Programmieren in Assembler

1.) Die folgende Programmbeschreibung ist für Anwender gedacht, die bereits einigermaßen mit dem 650x Befehlssatz und der650x ASSEMBLER-Sprache vertraut sind. Die nötigen Kenntnisse lassen sich durch Lektrüre der Firmendruckschrift "6502 Mikrocomputer Family Programming Manual" (deutsche oder englische Ausgabe; erhältlich bei MOSTECH. DEP. Frankfurterstraße 171 - 175 D-6078 Neu Isenburg) aneignen. Ferner wird auf die Seiten 91 ... 112, A 1.... A 9 des PET USER MANUAL (oktober 1978, englisch) sowie diverse Hinweise zum Gebrauch von USR und SYS in der PET Benutzer-Club - Lose Blattsammlung und die Dokumentation zum TIM Monitor verwiesen.

Die Programme sind verwendbar für PET 2001 8K mit dem Orginal-Betriebssystem, sowie die CBM 3001 Maschinen mit 16 oder 32K RAM.

Maschinensprache - Unterprogramme sind PET - komatibel)
Für die Bearbeitung von Programmen, die im Objektcode mehr als 200 Bytes umfassen, ist entweder Speichererweiterung über 8K hinaus oder ein zweiter Kassettenrecorder erforderlich. Mit Hilfe des Recorders können auch Programme bearbeitet werden, deren Ablagebereich in den vom ASSEMBLER-Programm belegten Speicher fällt.

2.) Sie finden auf Ihrer Kassette die 3 Programme EDITOR, ASSEMBLER und DISMON, sowie das Hilfsprogramm BINDER. EDITOR dient, ggf. zusammen mit BINDER, zum Erstellen und Korrigieren von Bändern, die Programme in Quell-(Assembler) Sprache enthalten (Q-Bänder), und als Eingangsdaten für den ASSEMBLER dienen. Dieser übersetzt die Quell-Programme in Objekt (Maschinen)-Code (und gibt ggf. Fehlermeldungen aus). Die Objekt-Programme können in den Speicher abgelegt oder auf Maschinen- (M-)Bändern aufgezeichnet werden - Letztere sind in einem Format, das von BASIC leicht und schnell gelesen werden kann, was die Einbindung von Maschinen- in BASIC-Programme erleichtert.
Der DISMON schließlich dient zum Austesten und Korrigieren. Er benutzt als Eingangsdaten M-Bänder oder vom ASSEMBLER abgelegte Programme und liefert seinerseits Objekt-Programme im Speicher oder M-Bänder.

Die Q-Bänder haben folgendes Format:

(1.Aufz.) , ENDE (2.Aufz.) . ENDE

Dabei stimmen (1.Aufz.) und (2.Aufz.) überein und bestehen aus Strings, die durch CHR$ (13) abgetrennt sind.

Für M-Bänder ist folgendes Format maßgeblich:

BOF Record1 Record2 ... Recordn Endzeichen EOF

Dabei besteht das Endzeichen aus den beiden Zeichen CHR $(1) CHR$(5) , und die Records, von denen beliebig viele möglich sind, tragen jeweils die Information für einen zusammenhängenden Speicherabschnitt. Sie beginnen mit dem Anfangszeichen
CHR$(1) CHR$(4) d...d CHR$(13), wobei d...d die String-Codierung die Zahl ist, die den Anfang des Abschnitts bezeichnet. Es folgt die Information für den Abschnitt bis zum nächsten Anfangs- oder zum Endzeichen. Sie ist folgendermaßen codiert:
CHR$(x) wenn x ≠ 0,1,10,29
CHR$(1) CHR$(x+1) in den Ausnahmefällen.
Diese Bänder lassen sich von dem folgenden einfachen BASIC-Programm laden:

```
10   OPEN1:FOR I=1TO 1E8:GET#1,A$:A=ASC(A$):
     IF A > 1THEN50
30   GET#1,A$:A=ASC(A$)-1:IF A=4 TEHN CLOSE 1:I=E9:
     NEXT:STOP
40   IF A=3THEN INPUT # 1,I:I=I-1:NEXT
50   POKE I,A:NEXT
```

(Vgl. auch das Listing von DISMON, wozu Debuggingzwecken u.a. noch die Statusabfrage eingebaut ist.)

3.) Beschreibung des EDITORS

Nach Laden und Starten meldet sich der EDITOR mit der Frage, welche Kassette zum Schreiben benutzt werden soll (Antwort 1 oder 2 gefolgt von RETURN; Sie sollten 2 aber nur wählen, wenn Kassette 2 angeschlossen ist). Anschließen wird ein Komma ausgegeben, vor dem der Cursor blinkt. Die Eingabe von Texten und Befehlen funktioniert im wesentlichen wie beim editieren von BASIC-Texten.
Eingeben: Zeilennr., Text RETURN
(im folgenden lassen wir stets RETURN weg!)
Der Text muß nur dann in Anführungszeichen eingeschlossen werden, wenn er Komma oder: enthält; zwischen Zeilennr. und Text muß genau ein Komma und dürfen beliebig viele Leerräume stehen. Eingabe von Zeilennr., allein löscht die betreffende Zeile. Als Zeilennr. sind alle natürlichen Zahlen bis 9999 zulässig; die Textzeile darf ohne Zeilennr. max.66 Zeichen umfassen. Wenn die

Zeilennr. fehlt, wird der Befehl ignoriert. Einschub von Zeilen ist jederzeit möglich, deshalb empfehlen sich für die Nummerierung 10er-Schritte.
Listen:

LI,	listet den ganzen Speicher
LI Zeilennr.,	listet genau diese Zeile
LI Zeilennr.1,Zeilennr.2	listet alle Zeilen mit dazwischen-liegenden Nummern
LI Zeilennr.,-	listet alles ab dieser Nummer
LI ;Zeilennr.	listet alles bis zu dieser Nummer.

Natürlich können Sie den Cursor in eine beliebige auf dem Schirm befindliche Zeile bringen, diese editieren und wieder eingeben. Benutzen Sie diese Funktionen um folgendes Quell-Programm zu erstellen:

```
 10,    *=830
 20,    ! TESTPROGRAMM
 30,    LDY # 0
 35,    LDX # 0
 40,    "MARKE LDA 32768, Y "
 50,    EOR # % 10000000
 60,    "MK1 STA 32768,Y"
 70,    INY
 80,    BNE MARKE
 90,    INC MARKE+2
100,    INC MK1+2
110,    INX
120,    CPX # 4
130,    BNE MARKE
140,    LDA # 32768>
150,    STA MARKE+2
160,    STA MK1+2
170,    RTS
```

Durch den Befehl
SA,
können Sie dieses Programm auf ein Q-Band übernehmen (zur teilweisen Übernahme kann man ähnlich wie beim Listen SA, Zeilennr., usw. verwenden; die Verifikation ist aber bei teilweiser Übernahme nicht möglich.) Die Maschine fordert Sie auf die Schreibkassette vorzubereiten und listet das Programm zweimal auf dem Schirm aus. Anschließend können Sie das Band durch

VE,	überprüfen (Meldungen:BAND OK bzw. FEHLER) und wenn Sie wollen, den Speicher durch
CL,	löschen.

Wenn Sie das Band zum weiteren Korrigieren wieder laden wollen, so ist dies durch

LO,	möglich.

Beim Einlesen werden die Zeilen automatisch auf 10er Abstände umnummeriert, wovon Sie sich durch LI, überzeugen können. DR*,* wirkt wie LI*,*, statt auf den Schirm wird jedoch auf einen Drucker ausgegeben, der als Gerät # 4 am IEEE - Bus angesprochen wird.
Um in einem längeren Programm eine bestimmte Zeile wiederzufinden, ist die Suchfunktion praktisch:

SU, Text	listet die erste Zeile in der Text vorkommt (in unserem Beispiel würde als Resultat von SU, EO etwa Zeile 50 gelistet),
SU Zeilennr., Text	listet die erste Zeile mit größerer Nummer als Zeilennr., in der Text vorkommt.

Wenn Text nirgends vorkommt, meldet das Programm "GIBT's NICHT". Die Funktion ist auch nützlich um Schriebfehler wie „ oder falsch geschriebene Symbole aufzuspüren.

FR,	gibt den noch verfügbaren Speicherplatz (er sollte 200 nicht unterschreiten).

Fehlerhafte Eingaben:
Doppeltes Komma verursacht EXTRA IGNORED, die Daten nach den „ werden nicht übernommen, fehlendes Komma verursacht ??, die Daten können dann noch eingegeben werden. Andere Befehle als die oben erwähnten ergeben ????, der Befehl wird ignoriert. Es sind max. 255 Zeilen möglich. Bei Überschreitung wird ?, PUFFER VOLL gemeldet. In diesem Fall erstelle man mehrere Q-Bänder und fasse sie mit Hilfe des BINDER's zu einem langen Band zusammen. Die nötigen Anweisungen sind im BINDER enthalten. Vor Laden des BINDER's Maschine abschalten! Der BINDER setzt die Existenz von Kassette 2 voraus.

Das Wort .ENDE darf außer nach ! nicht im Text vorkommen (vgl. 6 e). Wenn .ENDE als Text vorkommt, wird bei der Verifikation Fehlermeldung gegeben.

4.) Kurzbeschreibung ASSEMBLER

Schalten Sie vor Laden des Programms den PET aus und wieder ein (Begründung s.7). Laden Sie per RUN - das Programm fragt, ob das Quell-Programm vom Band (B) oder von DATA-Statements (D) stammt. Um letztere Möglichkeit wahrzunehmen, benötigen Sie bei längeren Quell-Programmen mehr als 8 K-Speicher. Sie können dann das Quell-Programm unter Zeilennummern ab 15.000 ablegen in der Form

Zeilennr.DATA TEXT,TEXT,...Zeilennr.DATA TEXT,... ...
Letzte Zeilennr.DATA TEXT, ..., .ENDE

(Das Programm wird in diesem Fall bis zum ersten Auftreten von . ENDE übersetzt; die einzelnen "Zeilen" werden durch Komma getrennt, "Zeilen" die "Komma" enthalten, müssen in " " gesetzt werden. Beispiel:

15000 DATA*=826,LDY #0,"M LDA L,Y", INY...

Anhängen der DATA-Zeilen sowie Listen des ganzen Programms ist erst möglich, wenn nach dem Laden RUN gegeben und dann gestoppt wurde (s.7)!)

Wenn Sie sich für "B" entschieden haben, werden Sie aufgefordert, die Kassette 1 vorzubereiten, etwa das Q-Band aus dem vorigen Abschnitt einzulegen. Danach fragt das Programm ob Mitschrieb (erstellen eines M-Bandes parallel zur Übersetzung) erwünscht ist. Diese Möglichkeit können und sollten Sie wahrnehmen wenn Sie eine zweite Kassette haben. (Drücken Sie "1", andernfalls "0"). Es folgt die Anfrage, ob das erstellte Objekt-Programm in den Speicher abgelegt werden soll. Dies kann nur geschehen, wenn der Ablagebereich nicht mit dem Programmbereich des ASSEMLER's zusammenfällt. Unser Beispiel ist in dieser Hinsicht harmlos, also geben Sie "1". Durch die folgende Anfrage Druck Druck 1/0 können Sie bestimmen, ob die Ausgabe des Assemblerprotokolls auf den Drucker (würde Gerät # 4) oder auf den Schirm protokolls auf den Drucker (würdeL4) oder auf den Schirm erfolgen soll. Nun beginnt die Übersetzung. Im ersten Durchlauf werden nur Fehlermeldungen (wenn nötig) ausgegeben, im zweiten Durchgang wird ein komplettes Protokoll erstellt. Die Form der Protokollzeilen ist so gewählt, daß Ausgabe auf jedem Drucker möglich ist. Danach wird eine Symbol-Tafel erstellt, die die Werte der verwendeten Symbole (in Hex.) enthält. Sie dient als Referenz und Kontrolle (vgl.Fehler 8). Damit ist die eingentliche Übersetzung abgeschlossen. Sie haben nun die Möglichkeit, Speicherinhalt als M-Band zu übernehmen. Wenn Sie keinen Mitschrieb erstellt haben, sollten Sie hier "1" geben - die Aufzeichnung erfolgt auf Kassette 1. Sie müssen dann angeben von/bis zu welcher

Lokation (dez.) aufgezeichnet werden soll. Nach Schreiben des Records wird zum Schreiben des nächsten aufgefordert bis Sie durch Eingabe von sinnlosen Lokationen das Band abschließen: Ist die zweite Lokation nämlich kleiner gleich der ersten, so wird das Endzeichen CHR$(1) CHR$(5) geschrieben und das Band abgeschlossen. (Ein Record umfaßt also wenigstens 2 Bytes.) Wenn Sie schließlich auf die "AUF BAND?" -Anfrage hin "O" gedrückt haben, können Sie das Musterprogramm durch SYS (830) starten. Es sollte alles, was sich auf dem Schirm befindet, in REVERSE FIELD verwandeln.

5.) Nun laden Sie den DISMON, um ihn zu testen. Wenn Sie zuvor den ASSEMBLER geladen hatten, ist Abschalten nicht erfoderlich (nicht ratsam, wenn das Objekt-Programm im Speicher steht).
Der DISMON meldet sich mit der Frage, ob Speicherreservierung am "oberen Ende" (vgl.7) gewünscht ist. Wenn ja, ("1") wird angefragt ab wo. Die Reservierung wird nur ausgeführt, wenn sie den Betrieb des DISMON nicht stört. Anschließend fordert das Programm, ähnlich wie der EDITOR, zur Eingabe von Befehlen auf.
Folgende Funktionen existieren:

Shhhhh,	wandelt die eingegeben Hex-Zahl in eine Dex-Zahl um.
dddddd,	wandelt die eingegeben Dex-Zahl in eine Hex-Zahl um.
LI Loc1, Loc2	listet den Speicher von Loc1 bis Loc2 einschließlich in "Disassembler" Form.
LI Loc1,	listet Lokation 1

Loc1 und Loc2 können dezimal oder hex. (mit vorgesetztem $!) gegeben werden.
Die "Disassembler" Form lehnt sich an die Form der Quell-Programme an, allerdings sind alle Operanden in Hex. ausgedrückt.
Die Zeilen haben im einzelnen das Format

Loc	Loc	max.3 Bytes Speicher	OPCODE	Operand
Dex	Hex	Hex	Mnemo	Hex

Bei Verzweigungsbefehlen (BEQ...) ist der Operand absolut angegeben.
PO Loc, Wert lädt die angegebne Lokation (Dez. oder $ Hex.) mit dem Wert; als Wert ist möglich:
1. Dez. (größer Ø)
2. $Hex. (auch$Ø)
3. AssemblerMnemo, gefolgt von einer Zahl zwischen Ø und 1Ø für den Adress-Modus (kann bei 1-Byte und Branch-Befehlen entfallen). Für die Modus-Codierung gilt: 0:(indir,X),1:zero page

2:immediate,3:absolute, 4(indir),Y ,5:zero page,X 6:absolute,Y ,7:absolute,X ,8:(indir),9:zero page,Y ,10:Akku
Beispiel:
PO$33A,LDA2
Sollte die Eingabe in irgendenem Pumkt unkorrekt sein, so werden die Modus-Codes dargestellt.
Mit dieser Funktion lassen sich leicht Programmkorrektruren ausführen.

RU Loc, Parameter	startet ein bei Loc ($Hex oder Dez) beginnendes M-Programm mit dem angegebenen USR-Parameter und gibt den Wert von USR aus.
EX,	Ausgang nach BASIC um ein am "unteren Ende" (vgl.7) abgelegtes Programm per SAVE auf ein BASIC-Programmband übernehmen zu können.

Befehle für den Umgang mit Bändern:

SA Loc1, Loc2	eröffnet ein M-Band und schreibt von Loc1 bis Loc2 (einschließlich). Danach wird ?SA , erzeugt; Übereinstimmung von erster und zweiter Lokation bei Eingabe schließt das Band ab.
VE,	vergleicht ein komplettes M-Band gegen den Speicher. Meldungen: BAND OK / FEHLER.
TE,	liest ein M-Band ein und gibt (in Dez !) an, welche Speicherbereiche von diesem Band belegt würden (es erfolgt keine Übernahme) Fehleranzeige: Wenn die Statusvariable ungleich 0 ist, wird sie ausgegeben (vgl. PET-Handbuch) und es erfolgt Abbruch.
LO Loc1,Loc2	lädt alles, was sich auf einem M-Band mit Lokationen zwischen Loc1 und Loc2 befindet, in den Speicher. Beide Lokationen müssen angegeben werden. Es wird, wenn nötig, Statusmeldung gegeben.

6.) Ausführliche Beschreibung des ASSEMBLER's:
a.) Das Befehlsformat für echte Operationen.
Zeilen mit echten Instruktionen sind solche, die als Befehle ins Objektprogramm übersetzt werden. Das Format ist
(LABEL) OPCODE OPERAND (!KOMMENTAR)

Eingeklammerte Bestandteile können entfallen. Alle Bestandteile müssen durch Zwischenräume getrennt sein.
Als LABEL (... marke) ist jedes Textsymbol (s.6 b) zulässig, aber ein und dasselbe Symbol darf nicht in mehreren Zeilen als LABEL vorkommen als LABEL ist das Symbol definiert, sein Wert ist die laufende Lokation, das ist die Lokation, auf die OPCODE abgelegt wird.
KOMMENTAR kann beliebiger Text sein, dessen erstes Zeichen ein ! sein muß.
Die Schreibweise der Kombination OPCODE OPERAND entspricht genau der im MOS 650X Programming Manual angegebenen: die Regeln seien hier kurz wiederholt:
Als OPCODE sind zulässig:
1-Byte-Befehle:
RTS, BRK, BIT, CLC, CLD, CLI, CLV, DEX, DEY, INX, INY, NOP, PHA, PHP, PLA, PLP, RTI, SEC, SED, SEI, TAX, TAY, TSX, TXA, TXS, TYA
Verzweigungsbefehle:
BEQ, BMI, BNE, BPL, BVC, BVS, BCC, BCS
Sowie
ADC, AND, CMP, EOR, LDA, ORA, SBC, STA,
ASL, CPX, CPY, DEC, INC, JMP, JSR, LDX, LDY, LSR, ROL, ROR, STX, STY,
Bezüglich der Schreibweise der Operanden gilt:
Der Operand entfällt für 1 Byte-Befehle.
Für Verzweigungsbefehle kann der Operand jede Adresse mit Wert zwischen +127 und -128 sein (bei Überschreitung des Bereichs wird Fehler 3 gemeldet). Bei den restlichen Befehlen bestimmt der Wert des Operanden ggf. ob zero page oder absolute-Adressierung gewählt wird. Für definierte Operanden wird, soweit dies möglich ist, automatisch zero page gewählt (vgl. auch 6 b).
Schreibweisen:

OPCODE (Adr,X)	:	indiziert indirekt *
OPCODE (Adr),Y	:	indirekt indiziert *
OPCODE Adr	:	unmittelbar *
OPCODE Adr	:	absolute oder zero page
OPCODE (Adr)	:	indirekt (nur JMP)
OPCODE Adr,X	:	zero page oder absolute indiziert
OPCODE Adr,Y	:	zero page oder absolute indiziert

*Bei den ersten drei Adressierungsarten muß die Adresse im zweiten Durchlauf Wert 255 haben; sonst wird Fehler 2 gemeldet.
Verstöße gegen obige syntaktische Regeln werden i.a. mit Fehler 0 honoriert.

b.) Ausdrücke/Symbole:

Als (Text) Symbole sind alle Kombinationen aus Buchstaben und Zahlen zulässig, die erstens mit einem Buchstaben beginnen und nicht mit einer Opcode-Abkürzung oder einem der Zeichen A,X,Y übereinstimmen und zweitens nicht mehr als 6 Zeichen umfassen. Ein Symbol wird definiert durch sein Auftreten als LABEL oder durch die Pseudoinstruktion SYMBOL = Ausdruck (s.6 c). Als Sondersymbol haben wir *, was die laufende Lokation bezeichnet.
Als Konstanten sind möglich:

ddddd max.5-stellige Dezimalzahlen
$hhhh max.4-stellige Hexzahlen
@OOOOO max.5-stellige Oktalzahlen
%..... max.16-stellige Binärzahlen
'A Einzelzeichen- der Wert ist ihr ASCII-Code.

Ausdrücke bestehen aus den 2-stelligen Operatoren +-*/, den 1-stelligen Operatoren > und < , sowie Textsymbolen, Konstanten und *. Die 2-stelligen Operatoren wirken grundsätzlich in der Reihenfolge ihres Auftretens, Klammern sind verboten. Der Operator < verwandelt den zur Linken stehenden Teilausdruck in das entsprechende "untere" Byte: 256 < = 0 und > liefert entsprechend das "obere" Byte: 256 > = 1.
Beispiele für die Werte von Ausdrücken:

2+7*10-1 =89
3*7+3/3 =8
4*64+2<+10 =12

Falls * gerade den Wert 100 hat: ***=10000
Ausdrücke können Werte zwischen 0 und 65535 haben; bei Überschreitung erfolgt Fehlermeldung (Fehler 2).
Ein Ausdruck heiße: Im ersten Durchlauf definiert ", wenn alle in ihm vorkommenden Symbole vor seinem Auftreten definiert sind (falls die vorkommenden Symbole durch die pseudooperation Symbol = Ausdruck eingeführt werden, ist diese Definition natürlich rekursiv zu interpretieren). Das Symbol * ist stets definiert; sein wechselnder Wert ist die laufende Lokation (1. Lokation der laufenden Zeile).
An dieser Stelle sei auf eine wesentliche Regel hingewiesen:
JEDER AUSDRUCK, DER ALS OPERAND VON BEFEHLEN (AUSSER VERZWEIGUNGSBEFEHLEN) VORKOMMT UND EINEN WERT < 256 HABEN SOLL, MUSS IM ERSTEN DURCHGANG DEFINIERT SEIN.
Andernfalls nimmt der ASSEMBLER absoluten Adressmodus an und reserviert im ersten Durchlauf 3 Byte für den fraglichen Befehl.

Da im zweiten Durchlauf unter Umständen zero page Modus (2 Byte) anwendbar ist, alle Symbole aber nach dem ersten Durchlauf feste werte haben, könnten einige Symbole unzutreffende Werte bekommen. Der ASSEMBLER vergleicht den Byte-Verbrauch im ersten und zweiten Durchgang und gibt bei Nichtübereinstimmung Fehlermeldung 8. Ein Vergleich der Symboltafel mit den Lokationen des Protokolls gibt dann Aufschluß, ob die Symbole noch korrekte Werte haben (die Werte der Symboltafel stammen aus dem ersten Durchgang).
Eine weitere Beschränkung ist durch die Länge der Symbol-Tabelle gegeben:
Es dürfen max. 250 Textsymbole verwendet werden. Bei Überschreitung erfolgt Fehlermeldung 1 jedesmal, wenn versucht wird, ein neues Symbol zu definieren. Diesselbe Meldung erfolgt auch beim Versuch ein und dasselbe Textsymbol mehrfach zu definieren.

c.) Die Pseudo-Operation "="
Diese Operation kommt in 2 Formen vor:
Textsymbol = Ausdruck
weist dem Symbol den Wert des Ausdrucks zu und
* = Ausdruck
definiert als laufende Lokation den Wert des Ausdrucks.
IN BEIDEN FÄLLEN GILT, DASS DER AUSDRUCK IM ERSTEN DURCHLAUF DEFINIERT SEIN MUSS.
(sonst erfolgt Fehlermeldung 7, bzw. 0)
(Mand bedenke die Zuweisung M1 = M1)
In Zeilen mit der "="-Pseudo-operation ist ! KOMMENTAR zulässig, Labels dagegen sind verboten.

d.) Die Pseudo-Operation .BYTE .ADRE .TEXT
Alle 3 Operationen dienen zum Ablegen von Daten in den Speicher.
Die Formate sind:
(LABEL) .BYTE Ausdr., Ausdr.,... Ausdr., (! KOMMENTAR)
(LABEL) .ADRE Ausdr.,Ausdr.,... Ausdr., (! KOMMENTAR)
(LABEL) . TEXTTEXT
Dabei entsprechen in der .BYTE-Operation die Ausdrücke Einzel-Bytes (0...255, bei Überschreitung Fehler 2), die nacheinander abgelegt werden.
Die Ausdrücke der .ADRE-Operation entsprechen 2-Byte-Werten (0...65535), die mit vertauschten "oberen" und "unteren" Bytes abgelegt werden (geeignet zum Erstellen von Adresslisten für indirekte Adressierung!);diese werden nacheinander abgelegt.
Beispiel:

.ADRE $01,$1FF ergibt im Speicher 01,00,FF,01.
.TEXT übernimmt alle rechts stehenden Zeichen (einschließlich führender Leerräume!) in der für Ausgabe auf den Bildschirm benötigten Codierung.(SHIFT ist erlaubt, REVERSE FIELD nicht!) Nach .TEXT ist das leere String nicht zulässig.
Die Operation dient zum Bereitstellen von Textmeldungen. Kommentar ist nicht zulässig, der als Text übernommen wird.

e.) Die .ENDE-Operation zeigt dem Programm an, daß das Ende des Quell-Textes erreicht ist. Sie muß eingefügt werden, wenn das Quell-Programm in Form von DATA-Statements an den ASSEMBLER angehänt wird. Bei Erstellung von Q-Bändern durch den EDITOR oder BINDER wird sie automatisch erzeugt und darf nicht in den Quell-Text eingefügt werden.

f.) Zeilen der Form
!KOMMENTAR
sind zulässig, sie dürfen aber nicht mit einem LABEL versehen werden.
Zeilen, die nur aus einem LABEL bestehen, sind unzulässig.

7.) Speicherbereiche für M-Programme, die mit BASIC verbunden werden sollen (PET 2001 8K)

Lokationen dez.	Verwendung	Zum Reservieren verstelle Zeiger	Lademöglichkeit (von BASIC)	Abänderungsmögl. BASIC	Abänderungsmögl. M-PRG	Bem.
826...1023 (Puffer 2)	kleine Programme, Variabeln		M-Band	beliebig	soweit Vorrat	
1024...**** vor BASIC	universell	122/123 (40/41)	zus. mit BASIC-Prog.	beliebig	beliebig (erweitern mögl.)	1
... zwischen BASIC u. Variabeln	universell	124/125 (42/43)	zus. mit BASIC-Prog.	unmöglich	beliebig	2
***...TOPRAM oberhalb der Strings	universell	132/133 (50/51) 134/135 (52/53)	M-Band	beliebig	beliebig	3

In Klammern gilt für CBM

Bemerkung 1
Nach Verstellen des Zeigers 122/123 (40/41) lädt BASIC LOAD noch jedes Band von Lokation 1024 an und BASIC SAVE schreibt ebenso von Lokation 1024 bis zum Ende des BASIC-Textes (Variablenbeginn); BASIC SAVE dagegen schreibt von Loc (40/41) bis zum Ende des BASIC-Textes Lediglich bei der Ausführung von Programm- und Kommandomodus befehlen (insbesondere LIST und Programmkorrektur) wird angenommen, daß das zweite Zeichen der ersten Zeile (Zeilenzeiger 10) in der durch 122/123 (40/41) gegebenen Lokation zu finden ist. Dieser Zeiger wird beim Laden von Bändern im allgemeinen nicht verstellt.
Um Speicherraum für ein M-Programm zu reservieren, gehe man wie folgt vor:
Nach Einschalten der Maschine gebe man die Zeile
1 POKE 122,(Ad.lo):POKE 123,(Ad.hi):RUN
ein und Poke 0 in die Lokation (Ad.lo) + 256*(Ad.hi) -1,+0,+1. Dann gebe man RUN. Nun ist der Bereich von 1050...(Ad.lo)+ 256*(Ad.hi)-2 für M-Programme geschützt. Das BASIC-Programm kann nun bliebig eingegeben und geändert werden. Das Laden der M-Programme erfolgt einmalig mittels des in Abschnitt 2 angegebenen Hilfsprogramms, von da an automatisch per LOAD zusammen mit dem BASIC-Programm. Zum Erweitern des M-Bereichs gebe man in das BASIC-Programm unter niedrigsten Zeilennummern REM-Zeilen ein und richte den Zeiger 122/123 (40/41) auf eine dieser Zeilen. Die oben erwähnte Hilfszeile ändert man durch POKE'n in den BASIC-Text in geeigneter Weise ab (sie beginnt bei Speicherplatz 1024).
Da der Zeiger beim Laden nicht verstellt wird, ist es erforderlich, vor dem Laden eines neuen Programms den Zeiger durch
POKE 122 (40),1:POKE 123 (41),4 zurückzustellen oder die Maschine abzuschalten.
In dieser Technik erstellte Programme lassen sich erst dann listen und ändern, wenn der Zeiger verstellt ist (was aufgrund der im Speicher verbliebenen Hilfszeile 1 durch RUN erfolgen kann). Zum SAVE'n auf Kassette muss zunächst der Anfangszeiger durch POKE 40,1:POKE 41,4 zurückgestellt werden.
Die eben beschriebene Methode wird von ASSEMBLER, DISMON und EDITOR benutzt. Um mit dem DISMON M-Programme für diesen Bereich bearbeiten zu können, ist der Bereich von 1050... 2879 zum Ablegen von Benutzer-Programmen freigehalten. Der

ASSEMBLER belegt dagegen alle Lokationen von 1024...8191.

Bemerkung 2
Dieser Bereich ist nur brauchbar, wenn das BASIC-Programm nicht mehr geändert werden soll. Sonst muß das M-Programm voll relozierbar sein. Bei Reservierung verstelle man Zeiger 124/125 (42/43). lade das Programm auf Band und lese wieder ein.

Bemerkung 3
Bei Reservierung muß sowohl der TOPRAM als auch der TOP-STRING zeiger umgestellt werden. Auch diese Positionen werden beim Laden nicht beeinflußt. Der DISMON führt die Reservierung auf Wunsch am Programmbeginn aus.

Abschließende Bemerkung
Bezüglich des Gebrauchs von USR ( ) und SYS sei auf die Commodore-Informationen verwiesen. Warnung:
Für PET 2001 erstellte Maschinenprogramme, die den Gleitkommaakku 1, den BASIC-Eingangspuffer und ROM-Routinen wie INTFLP FLPINT sowie die ROM-Druck-Routinen benutzen, sind nicht mit dem CBM-System kompatibel, da zahlreiche Adressen geändert werden müssen.

8.) Fehlermeldungen des ASSEMBLER's

Bei Fehlern wird die beanstandete Zeile ausgegeben, darunter ***FEHLER , die Codenummer und ein Pfeil, der auf die ungefähre (!) Position des Fehlers weist.
Die Übersetzung schreitet in fast allen Fällen fort; dabei wird die Byte-Reservierung in einer Weise vorgenommen, daß sich der Code i.A. (notfalls durch Auffüllen mittels NOP) reparieren läßt (Ausnahme: Fehler 8). Im einzelenen gilt:

Fehler 0: Syntaxfehler oder *=undefinierter Ausdruck
Fehler 1: doppelt definiertes Symbol (Bei 8K auch bei >250 Symbole)
Fehler 2: unzulässiger Wert eines Ausdrucks: negativ, oder (größer 255 in .BYTE Pseudo-Operation, bzw. als immediate-Operand, oder größer 65535)
Fehler 3: Bereichsüberschreitung bei relativer Adressierung (Verzweigungsbefehle)

Fehler 4: unzulässiger Adressmodus (z.B.: LDY Adr.,Y)
Fehler 5: unzulässige Pseudo-Operation
Fehler 6: Symbol nach dem ersten Durchlauf nicht definiert
Fehler 7: in Symbol=Ausdruck Ausdruck nicht definiert
Fehler 8: (wird erst nach Erstellung des Protokolls gegeben) verschiedener Byte-Verbrauch im ersten und zweiten Durchgang; weist auf Verstoß gegen die Operandenregel hin; die Byte-Differenz wird ausgegeben.

System-Kommandos EDITOR
Zeilennr., Text
LI Zeilennr.,
LI Zeilennr., Zeilennr.
LI Zeilennr.,-
LI , Zeilennr.
LO ,
VE ,
SA ,
SU , Text
SU Zeilennr., Text
CL ,
FR ,
DR Zeilennr.,
DR Zeilennr., Zeilennr.
DR Zeilennr.,-
DR , Zeilennr.
System-Kommandos DISMON
LI Loc. ,
LI Loc. , Loc.
SA Loc. , Loc.
LO Loc., Loc. (Lokationen müssen angegeben werden)
TE ,
VE ,
EX ,
PO Loc., Wert
PO Loc., MnemoModus
RU Loc., Parameter
$Hex ,
Dez ,

Zusammenstellung der Listings
für Commodore PET 8K mit alten ROM's

1. EDITOR

```
READY.

 1ØØ DIMT$(255),K$(3):PRINT"3"SPC(17Ø)"QUELLTEXT-EDITOR"
 1Ø5 POKE2,6:INPUT"SCHREIBKASS. 1/2";K:A=9999:K$(Ø)="LESEN"
:K$(1)="SCHREIBEN"
 1Ø7 K$(2)="BAND OK":K$(3)="FEHLER":Q$=CHR$(34)
 11Ø PRINT"3";:GOSUB8ØØØ
 15Ø PRINTSPC(5)",======";:INPUTN$,T$:N$=N$+"!!!"
 17Ø L$=LEFT$(N$,1):IFL$=" "ORL$="?"THENN$=RIGHT$(N$,LEN(N$
)-1):GOTO17Ø
 18Ø IFL$="!"THEN15Ø
 19Ø V=VAL(N$):IFV<1THENL$=LEFT$(N$,2):V=VAL(RIGHT$(N$,LEN(
N$)-2))
 2ØØ F=Ø:IFLEN(L$)=1THEN2ØØØ
 2Ø5 IFL$="FR"THENPRINT"? ,FREI"FRE(Ø):GOTO15Ø
 21Ø IFL$="CL"THEN11Ø
 22Ø IFL$="LO"THEN3ØØØ
 225 IFL$="VE"THEN7ØØØ
 23Ø IFL$="SA"THENN=1:F=1:GOSUB6ØØØ:GOTO26Ø
 24Ø IFL$="SU"THEN5ØØ
 25Ø IFL$<>"LI"THENPRINT"1???===2";:GOTO15Ø
 26Ø IFV>AORV<=ØTHENV=1
 27Ø N=V-1:V1=VAL(T$):IFV1<=ØTHENV1=V:IFT$="-"ORV=1THENV1=A
 28Ø POKE1,33:N=USR(N):IFN>V1ORN<ØTHEN4ØØ
 29Ø POKE1,135:L$=""
 31Ø PRINT"?";RIGHT$("    "+STR$(N),4);","Q$;T$(USR(N)):IFF=
ØTHEN28Ø
 32Ø PRINT#2,Q$+T$(USR(N))+Q$:Z=Z+LEN(T$(USR(N)))+5:IFZ<121
THEN28Ø
 33Ø POKEPO,PE:FORI=1TO1ØØ:NEXT:Z=Ø:POKEPO,PA:GOTO28Ø
 4ØØ IFF=ØTHEN15Ø
 41Ø PRINT#2,".ENDE":F=F+1:IFF=2THENN=V-1:GOTO28Ø
 42Ø CLOSE2:GOTO15Ø
 5ØØ IFV>ATHENV=Ø
 51Ø T=LEN(T$):IFT$=""THEN15Ø
 52Ø POKE1,33:V=USR(V):IFV<ØORV>ATHENPRINT" ? ,GIBT'S NICHT
":GOTO15Ø
```

```
530 POKE1,135:L$=T$(USR(V)):N=LEN(L$):IFN<TTHEN520
540 FORI=1TON-T+1:IFT$=MID$(L$,I,T)THENI=99
550 NEXTI:IFI>98THENN=V:V1=V:GOTO290
560 GOTO520
2000 IFV>ATHEN250
2010 IFT$=""THENF=1
2020 POKE1050,F:POKE1,162:N=USR(V):IFN<0ANDF=0THENPRINT"?
,PUFFER VOLL":GOTO150
2030 IFN>=0THENT$(N)=T$
2040 GOTO150
3000 GOSUB8000:N=0:GOSUB6100:POKE1,162:FORV=10TO2570STEP10
:INPUT#1,T$
3010 IFT$=".ENDE"THENCLOSE1:V=A:NEXTV:GOTO150
3020 T$(USR(V))=T$:NEXTV
6000 PO=59456:PE=207:PA=223:Z=58:T=3:IFK=2THEN6200
6100 PO=59411:PE=53:PA=61:Z=122:T=2
6200 PRINT"KASS.";T-1;"ZUM "K$(N)" VORBER. TASTE"
6300 GETL$:IFL$=""THEN6300
6400 POKE243,Z:POKE244,T:OPENN+1,T-1,N:Z=A:RETURN
7000 N=0:GOSUB6000:FORI=0TO1:FORV=0TOA:INPUT#1,L$:IFL$=".E
NDE"THEN7500
7010 POKE1,33:V=USR(V):IF(ST)ORV<0THENF=1:I=A:GOTO7500
7020 PRINTL$:POKE1,135:IFT$(USR(V))<>L$THENF=1
7030 V=V-1:NEXTV
7500 NEXTI:CLOSE1:PRINT"? ,"K$(F+2):GOTO150
8000 POKE1050,0:SYS(1685):FORI=0TO255:T$(I)="":NEXTI:RETUR
N
READY.
```

2. BINDER

```
READY.

 100 PRINT"3";SPC(248);"****** BINDER ******"
 110 PRINT:PRINT:PRINT:PRINT"KASS.2 VORBEREITEN (SCHREIBEN)
 ,TASTE
 120 DIMT$(255):GOSUB5000:Z=999:POKE243,58:POKE244,3:OPEN2,
2,1
 130 PRINT:PRINT:PRINT:INPUT"ZAHL DER ZU VERBINDENDEN BAEND
ER ";M:IFINT(M)<=0THEN130
 140 FORK=1TO2:FORI=1TOM
 150 GOSUB6000:T$(0)="! BAND"+STR$(I)
 155 FORL=1TO256:INPUT#1,T$:IFT$=".ENDE"THENL1=L:L=259:NEXT
L:CLOSE1:GOTO160
 157 T$(L)=T$:NEXTL
 160 FORL=0TOL1-1:IFZ<120THEN200
 170 POKE59456,207:FORJ=1TO150:NEXTJ:POKE59456,223:Z=0
 200 T$=CHR$(34)+T$(L)+CHR$(34):Z=Z+LEN(T$):PRINT#2,T$:PRIN
TT$
 220 NEXTL,I:PRINT#2,".ENDE"
 225 PRINT:PRINT"******ZWEITER DURCHLAUF":NEXTK:CLOSE2
 230 PRINT:PRINT:PRINT"VERIFIKATION ERWUENSCHT 1/0";:INPUTZ
:IFZ=0THENEND
 240 PRINT:PRINT:PRINT"KASS.2 RUECKSPULEN,TASTE":GOSUB5000:
POKE243,58:POKE244,3
 250 OPEN2,2,0:FORK=1TO2:FORI=1TOM:GOSUB6000
 260 INPUT#2,T$
 270 INPUT#1,V$:IFV$=".ENDE"THEN400
 280 INPUT#2,T$:PRINTT$:IFT$=V$THEN270
 290 PRINT"********* FEHLER *********":CLOSE1:CLOSE2:GOTO23
0
 400 CLOSE1:NEXTI:INPUT#2,T$:NEXTK:CLOSE2
 410 PRINT:PRINT:PRINT"********* BAND OK *********":END
 5000 GETA$:IFA$=""THEN5000
 5010 RETURN
 6000 PRINT:PRINT:PRINT"********* ZUM EINLESEN
 6005 PRINT"BAND"I"IN KASS.1 EINLEGEN,RUECK,TASTE"
 6010 GOSUB5000:POKE243,122:POKE244,2:OPEN1:RETURN
READY.
```

```
READY.

 2 GOTO5ØØ
 6 T$="?":IFP>LEN(L$)THENRETURN
 8 FORP=PTOLEN(L$):C$=MID$(L$,P,1):D=ASC(C$):IFD=32THENNEXT
P:RETURN
 1Ø IFD=33THENRETURN
 12 IFD=39ORD=36ORD=64ORD=37ORD>47ANDD<58THENT$=";":GOTO3Ø
 14 T$=C$:IFD<65ORD>9ØTHENP=P+1:RETURN
 16 H=P:C1=Ø:I1=Ø:FORI=1TO6:D=ASC(MID$(L$,P,1))-48
 18 IF(D<17ORD>42)AND(D<ØORD>9)THENI=7:GOTO22
 2Ø I1=43*I1+D:P=P+1
 22 NEXTI:IFP=H+1AND(C$="A"ORC$="X"ORC$="Y")THENRETURN
 24 T$=":":IFI1>68344ORI1<32312THENRETURN
 26 POKE2,1Ø:POKE1,246:H=USR(I1-49613):IFH<XTHENC=PEEK(1Ø52
):O=H:C1=-1
 28 POKE2,11:RETURN
 3Ø IFD=39THENN=ASC(MID$(L$,P+1,1)):P=P+2:RETURN
 32 N=D-48:H=4:H4=1Ø:IFD=36THENN=Ø:H4=16
 34 IFD=64THENH=5:N=Ø:H4=8
 36 IFD=37THENN=Ø:H=16:H4=2
 38 FORI=1TOH:P=P+1:L4=ASC(MID$(L$,P,1))-48:IFL4>9THENL4=L4
-7:IFL4<1ØTHENRETURN
 4Ø IFL4<ØORL4>=H4THENRETURN
 42 N=N*H4+L4:NEXTI:P=P+1:RETURN
 45 IFP1=1ORL7=ØTHENRETURN
 47 Z1=X:PRINT#2,CHR$(1);CHR$(M);L:RETURN
 5Ø IFP1=1THENRETURN
 52 IFL6=1THENPOKEL9,V9
 54 IFL7<>1THENRETURN
 56 IFV9=ØORV9=1ORV9=1ØORV9=29THENPRINT#2,CHR$(1);CHR$(V9+1
);:Z1=Z1+2:GOTO6Ø
 58 PRINT#2,CHR$(V9);:Z1=Z1+1
 6Ø IFZ1<17ØTHENRETURN
 62 POKEPO,PE:FORZ1=1TO1ØØ:NEXTZ1:POKEPO,PA:Z1=1:RETURN
 66 PRINT:GOSUB11ØØØ:PRINT"*****FEHLER";E9;SPC(P);"↑":E9=Ø:
RETURN
 68 IFT$=":"THENV9=I1:GOSUB115ØØ:A=H:F1=F:GOTO74
 7Ø IFT$<>";"ANDT$<>"*"THENA=Y:GOTO233Ø
 72 F1=Ø:A=N:IFT$="*"THENA=L
```

```
 74 GOSUB6:O$=T$:IFNOT(O$="+"ORT$="-"ORT$="*"ORT$="/")THEN9
2
 76 GOSUB6:V9=I1:IFT$=":"THENGOSUB11500:F1=(F)ORF1:L4=H:GOT
O82
 78 L4=L:IFT$=";"THENL4=N:GOTO82
 80 IFT$<>"*"GOTO2330
 82 IFO$="+"THENA=A+L4
 84 IFO$="-"THENA=A-L4
 86 IFO$="*"THENA=A*L4
 88 IFO$="/"THENA=INT(A/L4)
 90 GOTO74
 92 IFF1THENA=Y
 94 IFO$="<"THENA=A-Y*INT(A/Y):GOTO74
 96 IFO$=">"THENA=INT(A/Y):GOTO74
 98 IFP1=2ANDA>X2ORA<0THENE9=2:GOTO66
 99 RETURN
 110 S=C%(I)AND15:L4=(C%(I)-S)/16:IFS>9THENS=S+7
 120 IFL4>9THENL4=L4+7
 130 PRINTCHR$(L4+48);CHR$(S+48);:IFI=1THENPRINT" ";
 140 RETURN
 500 X=255:X1=65535:X2=X1:DIMC%(4):POKE2,11:POKE1055,0:B$="
33343344432"
 600 L=826:Y=256:INPUT"QUELLE D/B";Q$:IFQ$="B"THENM=1:GOSUB
13500:OPEN1
 610 INPUT"MITSCHR. 1/0";L7:IFL7=1THENM=2:GOSUB13500:OPEN2,
2,1:M=4:GOSUB47
 620 INPUT"ABLEGEN 1/0";L6
 1000 P1=1:E=0
 1005 PRINT:PRINT:L=826:FORL2=1TOX1:IFQ$="D"THENREADL$
 1010 IFQ$="B"THENINPUT#1,L$
 1020 P=1:L$=L$+"?":GOSUB2000:GOSUB9000:IFP1>1THENGOSUB1100
0
 1030 E=E+B*(2*P1-3):IFB>1THENL=L+B-1
 1050 IFP1=3THENRESTORE:PRINT:PRINTL2"ZEILEN"PEEK(1055)"SYM
BOLE":P1=2:GOTO1005
 1060 IFP1<4THENNEXTL2
 1065 CLOSE1:E=E-8:IFETHENE9=8:GOSUB66:PRINT:PRINT"DIFF. "E
 1070 IFPEEK(1055)=0THEN1500
 1100 PRINT:POKE1,214:FORP=0TOPEEK(1055)-1:F=USR(P):L$="":F
ORI=1TO6:H=INT(F/43)
 1110 L$=CHR$(F-43*H+48)+L$:IFH>0THENF=H:NEXTI
 1120 PRINTRIGHT$("        "+L$+" = ",10);:FORI=0TO1:C%(I)=P
EEK(1054-I)
```

```
 1150 GOSUB110:NEXTI:PRINTSPC(5);:NEXTP:PRINT:PRINT
 1500 M=5:GOSUB45:CLOSE2:INPUT"AUF BAND 1/0";B:IFB=0THENEND
 1510 M=1:GOSUB13500:L7=1:OPEN2,1,1
 1520 INPUT"VON,BIS";L,B:IFL>=BTHEN1500
 1530 M=4:GOSUB47:FORL9=LTOB:V9=PEEK(L9):GOSUB56:NEXTL9:GOT
O1520
 2000 B=1:N1=-1:GOSUB6:IFT$="?"THENRETURN
 2050 IFC1ORT$<>":"GOTO2300
 2200 V9=I1:GOSUB6:H=L:IFT$="="ANDP1>1THENB=0:GOSUB11500:S=
H:RETURN
 2210 IFT$<>"="THEN2240
 2220 L9=I1:GOSUB6:GOSUB68:V9=L9:H=A:S=A:B=0:GOSUB11700:IFN
OTF1THENRETURN
 2230 E9=7:GOTO66
 2240 IFP1=1THENGOSUB11700
 2250 IFT$="?"THENRETURN
 2300 IFT$="."THEN3000
 2310 IFT$<>"*"THEN2500
 2320 GOSUB6:IFT$="="THENGOSUB6:GOSUB68:L=A:IFNOTF1THENM=4:
GOTO45
 2330 E9=0:GOTO66
 2500 IFNOTC1ORT$<>":"THEN2330
 2510 N1=0:IFC>24THEN4000
 2530 RETURN
 3000 K$=MID$(L$,P,4):P=P+4:IFK$="BYTE"THENX2=X:S=1:GOTO310
0
 3030 IFK$="ADRE"THENS=2:GOTO3100
 3040 IFK$="TEXT"GOTO3500
 3050 IFK$="ENDE"THENP1=P1+2:RETURN
 3060 P=P-4:E9=5:GOSUB66
 3100 IFP1=1THEN3600
 3120 GOSUB6:GOSUB68:H=INT(A/Y):V9=A-Y*H:L9=L+B-1:GOSUB50:B
=B+1
 3130 IFB<5THENC%(B)=V9
 3140 IFS=2THENV9=H:L9=L9+1:GOSUB50:B=B+1:IFB<5THENC%(B)=V9
 3170 IFT$=","GOTO3100
 3180 X2=X1:IFT$="?"THENRETURN
 3190 GOTO66
 3500 IFP1=1THENB=B+LEN(L$)-P:RETURN
 3550 FORP=PTOLEN(L$)-1:T$=MID$(L$,P,1)
 3580 V9=(ASC(T$)AND63)+(ASC(T$)AND128)/2:L9=L+B-1:GOSUB50:
IFB<4THENC%(B+1)=V9
 3590 B=B+1:NEXTP:RETURN
```

4. DISASSEMBLER

```
READY.

 2 DATALO,VE,TE,RU,LI,PO,SA,"!!",EX:DIMK$(1):K$(0)="LESEN":
K$(1)="SCHREIBEN"
 3 B$="22232233321":GOTO1000
 5 H=N:H$=""
 6 H1=INT(H/16):H2=H-16*H1:IFH2>9THENH2=H2+7
 8 H$=CHR$(H2+48)+H$:IFH1>0THENH=H1:GOTO6
 10 IFLEN(H$)AND1THENH$="0"+H$
 12 RETURN
 15 V=VAL(M$):M$=M$+"!!!":IFV>0THENRETURN
 17 H$=LEFT$(M$,1):IFH$="?"ORH$=" "THENM$=RIGHT$(M$,LEN(M$)
-1):GOTO17
 18 IFLEFT$(M$,1)<>"$"THENRETURN
 20 FORI=2TO98:M=ASC(MID$(M$,I,1)):IFM<48OR(M<65ANDM>57)ORM
>70THENI=99:NEXTI:RETURN
 22 M=M-48:IFM>9THENM=M-7
 24 V=V*16+M:NEXTI
 1000 PRINT"3";SPC(209);"DISASSEMBLER/MONITOR";SPC(200):PRI
NT
 1010 INPUT"TOPRESERVIERUNG 1/0";T:PRINT:PRINT:IFT<>1THEN11
00
 1020 INPUT"AB";T:IFT>PEEK(134)+256*PEEK(135)ORT<5700THENPR
INT"SINNLOS":GOTO1020
 1030 N=(T)AND255:T=(T-N)/256:POKE134,N:POKE135,T:POKE132,N
:POKE133,T
 1100 PRINTSPC(8)",=========";:INPUTM$,T$:N=VAL(M$)
 1110 IFN>0THENGOSUB5:PRINT"  $"H$:GOTO1100
 1120 GOSUB15:IFV>0THENPRINT" "V:GOTO1100
 1130 H$=LEFT$(M$,2):RESTORE:T=0:FORI=1TO9:READX$:IFX$=H$TH
ENT=I
 1140 NEXTI:IFT=0THENPRINT"1????1":GOTO1100
 1150 M$=RIGHT$(M$,LEN(M$)-2):GOSUB15:V1=V:M$=T$:GOSUB15
 1160 ONTGOTO1500,1500,1500,3000,3500,4000,4500,1100
 1170 END
 1500 GOSUB7000:OPEN1:B=0:FORJ=0TO1E8:GET#1,X$:S=ST:X=ASC(X
$)
 1510 IF(S)THENPRINT:PRINT"STATUS"S:CLOSE1:J=1E9:NEXTJ:GOTO
1100
 1520 IFX>1THEN1600
```

```
 1530 GET#1,X$:X=ASC(X$)-1:IFX=4THENCLOSE1:PRINT"BIS";J-1:J
=1E9:NEXTJ:GOTO1100
 1540 IFX<>3THEN1600
 1545 IFJ<=BTHENPRINT"1":PRINT"            1"
 1547 IFJ>BTHENPRINT"BIS";J-1
 1550 INPUT#1,B:PRINT"VON"B;:J=B-1:NEXTJ
 1600 ONTGOTO1700,1800
 1610 NEXTJ
 1700 IFV>=JANDV1<=JTHENPOKEJ,X
 1710 NEXTJ
 1800 IFX<>PEEK(J)THENPRINT:PRINT"?**,FEHLER : LOC."J"B."X"
S."PEEK(J)
 1810 NEXTJ
 3000 IFV1=0THEN1100
 3010 X=INT(V1/256):POKE2,X:POKE1,V1-256*X:PRINT"RESULTAT";
USR(V):GOTO1100
 3500 POKE1,69:POKE2,11:IFV1>65535THEN1100
 3510 FORJ=V1TOV:X=INT(J/256):POKE11,J-256*X:POKE12,X:T=USR
(0)
 3520 PRINTRIGHT$("    "+STR$(J),5);:N=J:GOSUB5:PRINTRIGHT$(
"   $"+H$,6);"   ";
 3530 X=PEEK(2884):IFX<25ORT=-1THENB=1:GOTO3600
 3540 IFX<33THENB=2:GOTO3600
 3545 Z=PEEK(2880):IFZ>7ANDX<41THENZ=0
 3550 B=VAL(MID$(B$,Z+1,1))
 3600 FORK=1TOB:N=PEEK(2880+K):GOSUB5:PRINT" "H$;:NEXTK:PRI
NTSPC(13-3*B);
 3610 IFT=-1THEN3800
 3620 T=T+49613:H$="":FORK=1TO3:N=INT(T/43):H$=CHR$(48+T-43
*N)+H$:T=N:NEXTK
 3630 PRINTH$" ";:IFB=1ANDX<33THEN3800
 3640 IFB>2ORX>32THEN3700
 3650 X=PEEK(2882):IFX>127THENX=X-256
 3660 N=X+J+2:GOSUB5:PRINTH$;:GOTO3800
 3700 IFZ=10THENPRINT"A";:GOTO3800
 3710 IFZ=2THENPRINT"#";
 3720 IFZ=0ORZ=4ORZ=8THENPRINT"(";
 3730 N=PEEK(2882)+256*(B-2)*PEEK(2883):GOSUB5:PRINTH$;
 3740 IFZ=8THENPRINT")";
 3750 IFZ=4THENPRINT"),Y";
 3760 IFZ=0THENPRINT",X)";
 3770 IFZ=6ORZ=9THENPRINT",Y";
 3780 IFZ=7ORZ=5THENPRINT",X";
```

```
 3600 FORP=PTOLEN(L$):IFMID$(L$,P,1)=","THENB=B+S
 3610 NEXTP:B=B+S:X2=X1:RETURN
 4000 GOSUB6:IFT$="A"THENM=10:RETURN
 4050 IFT$="#"THENM=2:GOSUB6:X2=X:GOSUB68:X2=X1:RETURN
 4110 IFT$<>"("GOTO4500
 4120 GOSUB6:GOSUB68:IFT$<>","THEN4300
 4160 M=0:IFA>XTHENE9=2:GOSUB66
 4170 GOSUB6:IFT$="X"THENGOSUB6:IFT$=")"THENRETURN
 4180 GOTO2330
 4300 M=4:IFT$<>")"THEN2330
 4320 GOSUB6:IFT$="?"THENM=8:RETURN
 4400 IFA>XTHENE9=2:GOSUB66
 4410 IFT$=","THENGOSUB6:IFT$="Y"THENRETURN
 4420 GOTO2330
 4500 GOSUB68:IFT$<>"?"THEN4600
 4520 M=1:IFA>XTHENM=3
 4540 RETURN
 4600 IFT$<>","GOTO4520
 4620 GOSUB6:IFT$<>"X"GOTO4700
 4640 M=5:IFA>XTHENM=7
 4650 RETURN
 4700 IFT$<>"Y"GOTO4520
 4720 M=9:IFA>XTHENM=6
 4800 RETURN
 9000 IFN1THENRETURN
 9020 B=VAL(MID$(B$,M+1,1))
 9050 C%(4)=INT(A/Y):C%(3)=A-Y*C%(4):IFC<25THENC%(2)=0:B=2:
GOTO9800
 9060 IFC<33THEN9300
 9070 IFC<41THEN9200
 9080 POKE1,33:O=USR(M):IFO=XTHENE9=4:B=4:GOSUB66
 9090 C%(2)=0:GOTO9800
 9200 C%(2)=O:H=A-L-2:C%(3)=HANDX:B=3
 9230 IF(H>127ORH<-128)ANDP1=2THENE9=3:GOSUB66
 9240 GOTO9800
 9300 IFM=9THENM=6:B=B+1
 9310 IFM>7THENE9=4:GOSUB66
 9320 C%(2)=O+4*M:IFM=2ANDO=129THENB=4:E9=4:GOSUB66
 9800 FORI=2TOB:L9=L+I-2:V9=C%(I):GOSUB50:NEXTI:RETURN
 11000 L4=L:H=B:IFB=0THENL4=S:H=1
 11065 IFB>4THENH=4
 11070 C%(0)=INT(L4/Y):C%(1)=L4-Y*C%(0):FORI=0TOH:GOSUB110:
NEXTI
```

```
 11110 PRINTRIGHT$("            "+STR$(L2),13-2*H);" ";LEF
T$(L$,LEN(L$)-1)
 11120 RETURN
 11500 POKE1,147:F=USR(V9):H=Y:IFFANDP1>1THENE9=6:GOTO66
 11510 H=(1+F)*(PEEK(1053)+Y*PEEK(1054))-F*H:RETURN
 11700 L4=INT(H/Y):POKE1054,L4:POKE1053,H-Y*L4:POKE1,81
 11710 IFUSR(V9)THENE9=1:GOTO66
 11720 RETURN
 13500 PRINT"KASS."M"VORBER.,TASTE."
 13510 GETT$:IFT$=""THEN13510
 13520 Z1=X:PO=59411:PE=53:PA=61:POKE243,122:POKE244,2
 13530 IFM=2THENPO=PO+45:PE=239:PA=X:POKE243,58:POKE244,3
 13540 RETURN
READY.
```

```
3800 PRINT:J=J+B-1:NEXTJ:GOTO1100
4000 IFV1>65535THENPRINT"1????1":GOTO1100
4010 IFV>0ORLEFT$(M$,1)="$"THENPOKEV1,V:GOTO1100
4020 V2=VAL(RIGHT$(M$,LEN(M$)-3)):T=0:FORI=1TO3:X=ASC(MID$
(M$,I,1))
4030 T=T*43+X-48:NEXTI:POKE2880,V2:T=T-49613
4040 IFT<-17301ORT>18731THEN4200
4050 POKE1,150:POKE2,11:T=USR(T)
4060 IFT>-1THENPOKEV1,T:GOTO1100
4200 PRINT"??2MODI:0=(*,X) 1=ZE 2=# 3=AB 4=(*),Y"
4210 PRINT"5=ZE,X 6=AB,Y 7=AB,X 8=(*) 9=ZE,Y 10=AKK"
4220 GOTO1100
4500 IFV1=VANDFL=1THENPRINT#1,CHR$(1);CHR$(5):CLOSE1:FL=0:
GOTO1100
4510 IFV1=VANDFL=0THEN1100
4520 IFFL=0THENGOSUB7000:FL=1:OPEN1,1,1
4530 PRINT#1,CHR$(1);CHR$(4);V1:FORJ=V1TOV
4540 IFZ<150THEN4600
4550 POKE59411,53:FORJ1=0TO100:NEXTJ1:POKE59411,61:Z=0
4600 X=PEEK(J):IFX=0ORX=1ORX=10ORX=29THENPRINT#1,CHR$(1);:
X=X+1:Z=Z+1
4610 PRINT#1,CHR$(X);:Z=Z+1:NEXTJ
4620 PRINT"? SA000 ,0001":GOTO1100
7000 PRINT"KASS. 1 ZUM "K$(1+SGN(T-7))" VORBEREITEN;TASTE"
7010 GETX$:IFX$=""THEN7010
7020 POKE243,122:POKE244,2:Z=999:RETURN
READY.
```

Zusammenstellung der Listings
für Commodore CBM 16K und 32K mit neuen ROM's

1. EDITOR

```
READY.


 10 Y=256:C1=1050:C2=C1+1:C3=C1+2:DEFFNV(X)=PEEK(C1)+Y*PEEK
(C2)
 12 C4=1571:C5=1676:C6=1703:GOTO100
 30 X1=INT(X/Y):POKEC1,X-Y*X1:POKEC2,X1:RETURN
 100 DIMT$(255),K$(3):PRINT"3"SPC(170)"QUELLTEXT-EDITOR"
 105 INPUT"SCHREIBKASS. 1/2";K:A=9999:K$(0)="LESEN":K$(1)="
SCHREIBEN"
 107 K$(2)="BAND OK":K$(3)="FEHLER":Q$=CHR$(34)
 110 PRINT"3";:GOSUB8000
 150 PRINTSPC(5)",======";:INPUTN$,T$:N$=N$+"!!!"
 170 L$=LEFT$(N$,1):IFL$=" "ORL$="?"THENN$=RIGHT$(N$,LEN(N$
)-1):GOTO170
 180 IFL$="!"THEN150
 190 V=VAL(N$):IFV<1THENL$=LEFT$(N$,2):V=VAL(RIGHT$(N$,LEN(
N$)-2))
 200 F=0:IFLEN(L$)=1THEN2000
 205 IFL$="FR"THENPRINT"? ,FREI"FRE(0):GOTO150
 210 IFL$="CL"THEN110
 220 IFL$="LO"THEN3000
 225 CH=3:IFL$="VE"THEN7000
 230 IFL$="SA"THENN=1:F=1:T=K:GOSUB6000:GOTO260
 240 IFL$="SU"THEN500
 245 IFL$="DR"THENCH=4:GOTO260
 250 IFL$<>"LI"THENPRINT"1???===2";:GOTO150
 260 IFV>AORV<=0THENV=1
 270 N=V-1:V1=VAL(T$):IFV1<=0THENV1=V:IFT$="-"ORV=1THENV1=A
 275 OPEN3,CH
 280 X=N:GOSUB30:SYS(C4):N=FNV(0):IFN>V1THEN400
 300 X=N:GOSUB30:SYS(C5):X=FNV(0)
 310 PRINT#3,RIGHT$("    "+STR$(N),5);","Q$;T$(X):IFF=0THEN
280
 320 PRINT#2,Q$+T$(X)+Q$:GOTO280
 400 IFF=0THENCLOSE3:GOTO150
```

```
 410 PRINT#2,".ENDE":F=F+1:IFF=2THENN=V-1:GOTO280
 420 CLOSE2:CLOSE3:GOTO150
 500 IFV>ATHENV=0
 510 T=LEN(T$):IFT$=""THEN150
 520 X=V:GOSUB30:SYS(C4):V=FNV(0):IFV>ATHENPRINT" ? ,GIBT'S
 NICHT":GOTO150
 530 X=V:GOSUB30:SYS(C5):L$=T$(FNV(0)):N=LEN(L$):IFN<TTHEN5
20
 540 FORI=1TON-T+1:IFT$=MID$(L$,I,T)THENI=99
 550 NEXTI:IFI>98THENN=V:V1=V:L$="":OPEN3,3:GOTO300
 560 GOTO520
 2000 IFV>ATHEN250
 2010 IFT$=""THENF=1
 2020 POKEC3,F:X=V:GOSUB30:SYS(C6):N=PEEK(C1)
 2025 IFPEEK(C2)ANDF=0THENPRINT"? ,PUFFER VOLL":GOTO150
 2030 IFPEEK(C2)=0THENT$(N)=T$
 2040 GOTO150
 3000 GOSUB8000:N=0:T=1:GOSUB6000:FORV=10TO2570STEP10:INPUT
#1,T$
 3010 IFT$=".ENDE"THENCLOSE1:V=A:NEXTV:GOTO150
 3020 X=V:GOSUB30:SYS(C6):T$(FNV(0))=T$:NEXTV
 6000 PRINT"KASS.";T;"ZUM "K$(N)" VORBER. TASTE"
 6300 GETL$:IFL$=""THEN6300
 6400 OPENN+1,T,N:RETURN
 7000 N=0:T=K:GOSUB6000:FORI=0TO1:FORV=0TOA:INPUT#1,L$:IFL$
=".ENDE"THEN7500
 7010 X=V:GOSUB30:SYS(C4):V=FNV(0):IF(ST)ORV>ATHENF=1:I=A:G
OTO7500
 7020 PRINTL$:X=V:GOSUB30:SYS(C5):IFT$(FNV(0))<>L$THENF=1
 7030 V=V-1:NEXTV
 7500 NEXTI:CLOSE1:PRINT"? ,"K$(F+2):GOTO150
 8000 POKEC3,0:SYS(1690):FORI=0TO255:T$(I)="":NEXTI:RETURN
READY.
```

2. BINDER

```
READY.

 100 PRINT"3";SPC(248);"###### BINDER ######"
 110 PRINT:PRINT:PRINT:PRINT"KASS.2 VORBEREITEN (SCHREIBEN)
 ,TASTE
 120 DIMT$(255):GOSUB5000:OPEN2,2,1
 130 PRINT:PRINT:PRINT:INPUT"ZAHL DER ZU VERBINDENDEN BAEND
ER ";M:IFINT(M)<=0THEN130
 140 FORK=1TO2:FORI=1TOM
 150 GOSUB6000:T$(0)="! BAND"+STR$(I)
 155 FORL=1TO256:INPUT#1,T$:IFT$=".ENDE"THENL1=L:L=259:NEXT
L:CLOSE1:GOTO160
 157 T$(L)=T$:NEXTL
 160 FORL=0TOL1-1
 200 T$=CHR$(34)+T$(L)+CHR$(34):Z=Z+LEN(T$):PRINT#2,T$:PRIN
TT$
 220 NEXTL,I:PRINT#2,".ENDE"
 225 PRINT:PRINT"######ZWEITER DURCHLAUF":NEXTK:CLOSE2
 230 PRINT:PRINT:PRINT"VERIFIKATION ERWUENSCHT 1/0";:INPUTZ
:IFZ=0THENEND
 240 PRINT:PRINT:PRINT"KASS.2 RUECKSPULEN,TASTE":GOSUB5000
 250 OPEN2,2,0:FORK=1TO2:FORI=1TOM:GOSUB6000
 260 INPUT#2,T$
 270 INPUT#1,V$:IFV$=".ENDE"THEN400
 280 INPUT#2,T$:PRINTT$:IFT$=V$THEN270
 290 PRINT"######### FEHLER #########":CLOSE1:CLOSE2:GOTO23
0
 400 CLOSE1:NEXTI:INPUT#2,T$:NEXTK:CLOSE2
 410 PRINT:PRINT:PRINT"######### BAND OK #########":END
 5000 GETA$:IFA$=""THEN5000
 5010 RETURN
 6000 PRINT:PRINT:PRINT"######### ZUM EINLESEN
 6005 PRINT"BAND"I"IN KASS.1 EINLEGEN,RUECK,TASTE"
 6010 GOSUB5000:OPEN1:RETURN
READY.
```

```
READY.

 2 GOTO5ØØ
 6 C1=Ø:T$="?":IFP>LEN(L$)THENRETURN
 8 FORP=PTOLEN(L$):C$=MID$(L$,P,1):D=ASC(C$):IFD=32THENNEXT
P:RETURN
 1Ø IFD=33THENRETURN
 12 IFD=39ORD=36ORD=64ORD=37ORD>47ANDD<58THENT$=";":GOTO3Ø
 14 T$=C$:IFD<65ORD>9ØTHENP=P+1:RETURN
 16 FORI=ØTO5:D=ASC(MID$(L$,P,1)):IF(D<65ORD>9Ø)AND(D<48ORD
>57)THENI=7:GOTO22
 2Ø POKER+I,D:P=P+1
 22 NEXTI:SYS(3227):D=PEEK(R+13):IFD>ØTHENT$=CHR$(D):RETURN
 24 T$=":":C=PEEK(R+12):D=PEEK(R+6):IFD<XTHENO=D:C1=-1:F=-1
:RETURN
 26 F=(PEEK(R+7)=Ø):RETURN
 3Ø IFD=39THENN=ASC(MID$(L$,P+1,1)):P=P+2:RETURN
 32 N=D-48:H=4:H4=1Ø:IFD=36THENN=Ø:H4=16
 34 IFD=64THENH=5:N=Ø:H4=8
 36 IFD=37THENN=Ø:H=16:H4=2
 38 FORI=1TOH:P=P+1:L4=ASC(MID$(L$,P,1))-48:IFL4>9THENL4=L4
-7:IFL4<1ØTHENRETURN
 4Ø IFL4<ØORL4>=H4THENRETURN
 42 N=N*H4+L4:NEXTI:P=P+1:RETURN
 45 IFP1=1ORL7=ØTHENRETURN
 47 PRINT#2,CHR$(1);CHR$(M);L:RETURN
 5Ø IFP1=1THENRETURN
 52 IFL6=1THENPOKEL9,V9
 54 IFL7<>1THENRETURN
 56 IFV9=ØORV9=1ORV9=1ØORV9=29THENPRINT#2,CHR$(1);CHR$(V9+1
);:RETURN
 58 PRINT#2,CHR$(V9);:RETURN
 66 PRINT:GOSUB11ØØØ:PRINT#3,"*****FEHLER";E9;SPC(P);"↑":E9
=Ø:RETURN
 68 IFT$=":"THENGOSUB115ØØ:A=H:F1=F:GOTO74
 7Ø IFT$<>";"ANDT$<>"*"THENA=Y:GOTO66
 72 F1=Ø:A=N:IFT$="*"THENA=L
 74 GOSUB6:G=ASC(T$):IFNOT(G=43ORG=45ORG=42ORG=47)THEN92
 76 GOSUB6:IFT$=":"THENGOSUB115ØØ:F1=(F)ORF1:L4=H:GOTO82
 78 L4=L:IFT$=";"THENL4=N:GOTO82
 8Ø IFT$<>"*"GOTO66
```

```
82 IFG=43THENA=A+L4:GOTO74
84 IFG=45THENA=A-L4:GOTO74
86 IFG=42THENA=A*L4:GOTO74
88 A=INT(A/L4):GOTO74
92 IFF1THENA=Y
94 IFG=60THENA=A-Y*INT(A/Y):GOTO74
96 IFG=62THENA=INT(A/Y):GOTO74
98 IFP1=2ANDA>X2ORA<0THENE9=2:A=0:GOTO66
99 RETURN
110 D=C%(I)AND15:L4=(C%(I)-D)/16:IFD>9THEND=D+7
120 IFL4>9THENL4=L4+7
130 PRINT#3,CHR$(L4+48);CHR$(D+48);:IFI=1THENPRINT#3," ";
140 RETURN
150 B=B+1:GOSUB50:IFB<5THENC%(B)=V9
152 RETURN
500 R=1052:X=255:X1=65535:X2=X1:DIMC%(4):SYS(3203):B$="333
43344432"
600 L=826:Y=256:INPUT"QUELLE D/B";Q$:IFQ$="B"THENM=1:GOSUB
13500:OPEN1
610 INPUT"MITSCHR. 1/0";L7:IFL7=1THENM=2:GOSUB13500:OPEN2,
2,1:M=4:GOSUB47
620 INPUT"ABLEGEN 1/0";L6:INPUT"DRUCK 1/0";M:E=3:IFM=1THEN
E=4
1000 OPEN3,E:P1=1:E=0
1005 PRINT:PRINT:L=826:FORL2=1TOX1:IFQ$="D"THENREADL$
1010 IFQ$="B"THENINPUT#1,L$
1020 P=1:L$=L$+"?":GOSUB2000:GOSUB9000:IFP1>1THENGOSUB1100
0
1030 E=E+B*(2*P1-3):IFB>1THENL=L+B-1
1050 IFP1=3THENRESTORE:PRINT#3:PRINT#3,L2"ZEILEN"PEEK(R+11
)"SYMBOLE":P1=2:GOTO1005
1060 IFP1<4THENNEXTL2
1065 CLOSE1:E=E-8:IFETHENE9=8:GOSUB66:PRINT#3:PRINT#3,"DIF
F. "E
1070 PRINT#3:G=PEEK(R+11):IFG=0THEN1500
1100 FORP=1074TO1073+G:FORB=0TO5*250STEP250:PRINT#3,CHR$(P
EEK(P+B));
1120 NEXTB:PRINT#3," = ";:FORI=0TO1:C%(I)=PEEK(P+1750-250*
I)
1130 GOSUB110:NEXTI:PRINT#3,SPC(6);:NEXTP:PRINT#3:PRINT#3
1500 CLOSE3:M=5:GOSUB45:CLOSE2:INPUT"AUF BAND 1/0";B:IFB=0
THENEND
1510 M=1:GOSUB13500:L7=1:OPEN2,1,1
```

```
 1520 INPUT"VON,BIS";L,B:IFL>=BTHEN1500
 1530 M=4:GOSUB47:FORL9=LTOB:V9=PEEK(L9):GOSUB56:NEXTL9:GOT
O1520
 2000 N1=1:B=1:POKER+10,1:GOSUB6:IFT$="?"THENRETURN
 2050 IFC1ORT$<>":"GOTO2300
 2200 GOSUB6:H=L:IFT$="="ANDP1>1THENB=0:GOSUB11500:S=H:RETU
RN
 2205 IFP1=1ANDNOTFTHENE9=1:GOSUB66:GOTO2300
 2210 IFT$<>"="THEN2240
 2220 GOSUB6:GOSUB68:H=A:S=A:B=0:IFF1THENE9=7:GOSUB66
 2240 IFP1=1THEND=INT(H/Y):POKER+8,H-Y*D:POKER+9,D:SYS(3084
)
 2250 IFT$="?"THENRETURN
 2300 IFT$="."THEN3000
 2310 IFT$<>"*"THEN2500
 2320 GOSUB6:C1=0:IFT$="="THENGOSUB6:GOSUB68:L=A:IFNOTF1THE
NM=4:GOTO45
 2500 IFNOTC1THEN66
 2510 N1=0:IFC>24THEN4000
 2530 RETURN
 3000 C$=MID$(L$,P,4):P=P+4:IFC$="BYTE"THENX2=X:S=1:GOTO310
0
 3030 IFC$="ADRE"THENS=2:GOTO3100
 3040 IFC$="TEXT"GOTO3500
 3050 IFC$="ENDE"THENP1=P1+2:RETURN
 3060 P=P-4:E9=5:GOSUB66
 3100 IFP1=1THEN3600
 3120 GOSUB6:GOSUB68:H=INT(A/Y):V9=A-Y*H:L9=L+B-1:GOSUB150
 3140 IFS=2THENV9=H:L9=L9+1:GOSUB150
 3170 IFT$=","GOTO3120
 3180 X2=X1:IFT$="?"THENRETURN
 3190 GOTO66
 3500 IFP1=1THENB=B+LEN(L$)-P:RETURN
 3550 FORP=PTOLEN(L$)-1:G=ASC(MID$(L$,P,1))
 3580 V9=(GAND63)+(GAND128)/2:L9=L+B-1:GOSUB150:NEXTP:RETUR
N
 3600 FORP=PTOLEN(L$):IFMID$(L$,P,1)=","THENB=B+S
 3610 NEXTP:B=B+S:X2=X1:RETURN
 4000 GOSUB6:IFT$="A"THENM=10:RETURN
 4050 IFT$="#"THENM=2:GOSUB6:X2=X:GOSUB68:X2=X1:RETURN
 4110 IFT$<>"("GOTO4500
 4120 GOSUB6:GOSUB68:IFT$<>","THEN4300
 4160 M=0:IFA>XTHENE9=2:GOSUB66
```

```
 4170 GOSUB6:IFT$="X"THENGOSUB6:IFT$=")"THENRETURN
 4180 GOTO66
 4300 M=4:IFT$<>")"THEN66
 4320 GOSUB6:IFT$="?"THENM=8:RETURN
 4400 IFA>XTHENE9=2:GOSUB66
 4410 IFT$=","THENGOSUB6:IFT$="Y"THENRETURN
 4420 GOTO66
 4500 GOSUB68:G=(A>X):IFT$=","THEN4620
 4520 M=1-2*G:RETURN
 4620 GOSUB6:IFT$="X"THENM=5-2*G:RETURN
 4700 IFT$="Y"THENM=9+3*G:RETURN
 4720 GOSUB66:GOTO4520
 9000 IFN1THENRETURN
 9050 B=VAL(MID$(B$,M+1,1)):C%(4)=INT(A/Y):C%(3)=A-Y*C%(4):
IFC<25THENB=2:GOTO9800
 9060 IFC<33THEN9200
 9070 IFC<41THEN9300
 9080 POKER+8,M:SYS(3144):O=PEEK(R+6):IFO=XTHENE9=4:B=4:GOS
UB66
 9090 GOTO9800
 9200 H=A-L-2:C%(3)=HANDX:B=3
 9230 IF(H>127ORH<-128)ANDP1=2THENE9=3:GOSUB66
 9240 GOTO9800
 9300 IFM=9THENM=6:B=4
 9310 IFM>7THENE9=4:GOSUB66
 9320 O=O+4*M:IFO=137THENB=4:E9=4:GOSUB66
 9800 C%(2)=O:FORI=2TOB:L9=L+I-2:V9=C%(I):GOSUB50:NEXTI:RET
URN
 11000 L4=L:H=B:IFB=0THENL4=S:H=1
 11065 IFB>4THENH=4
 11070 C%(0)=INT(L4/Y):C%(1)=L4-Y*C%(0):FORI=0TOH:GOSUB110:
NEXTI
 11110 PRINT#3,RIGHT$("                    "+STR$(L2),13-2*H);LEFT
$(" "+L$,LEN(L$)):RETURN
 11500 H=Y:IFFANDP1>1THENE9=6:GOTO66
 11510 H=PEEK(R+8)+Y*PEEK(R+9)-F:RETURN
 13500 PRINT"KASS."M"VORBER.,TASTE."
 13510 GETT$:IFT$=""THEN13510
 13540 RETURN
READY.
```

4. DIASSEMBLER

```
READY.

 3 DATALO,VE,TE,RU,LI,PO,SA,"!!",EX:DIMK$(1):K$(Ø)="LESEN":
K$(1)="SCHREIBEN"
 4 B$="22232233321":C1=3121:IN=3134:GOTO1ØØØ
 5 H=N:H$=""
 6 H1=INT(H/16):H2=H-16*H1:IFH2>9THENH2=H2+7
 8 H$=CHR$(H2+48)+H$:IFH1>ØTHENH=H1:GOTO6
 1Ø IFLEN(H$)AND1THENH$="Ø"+H$
 12 RETURN
 15 V=VAL(M$):M$=M$+"!!!":IFV>ØTHENRETURN
 17 H$=LEFT$(M$,1):IFH$="?"ORH$=" "THENM$=RIGHT$(M$,LEN(M$)
-1):GOTO17
 18 IFLEFT$(M$,1)<>"$"THENRETURN
 2Ø FORI=2TO98:M=ASC(MID$(M$,I,1)):IFM<48OR(M<65ANDM>57)ORM
>7ØTHENI=99:NEXTI:RETURN
 22 M=M-48:IFM>9THENM=M-7
 24 V=V*16+M:NEXTI
 1ØØØ PRINT"3";SPC(2Ø9);"DISASSEMBLER/MONITOR";SPC(2ØØ):PRI
NT
 1Ø1Ø INPUT"TOPRESERVIERUNG 1/Ø";T:PRINT:PRINT:IFT<>1THEN11
ØØ
 1Ø2Ø INPUT"AB";T:IFT>PEEK(52)+256*PEEK(53)ORT<65ØØTHENPRIN
T"SINNLOS":GOTO1Ø2Ø
 1Ø3Ø N=(T)AND255:T=(T-N)/256:POKE5Ø,N:POKE51,T:POKE52,N:PO
KE53,T
 11ØØ PRINTSPC(8)",=========";:INPUTM$,T$:N=VAL(M$)
 111Ø IFN>ØTHENGOSUB5:PRINT"  $"H$:GOTO11ØØ
 112Ø GOSUB15:IFV>ØTHENPRINT" "V:GOTO11ØØ
 113Ø H$=LEFT$(M$,2):RESTORE:T=Ø:FORI=1TO9:READX$:IFX$=H$TH
ENT=I
 114Ø NEXTI:IFT=ØTHENPRINT"1????1":GOTO11ØØ
 115Ø M$=RIGHT$(M$,LEN(M$)-2):GOSUB15:V1=V:M$=T$:GOSUB15
 116Ø ONTGOTO15ØØ,15ØØ,15ØØ,3ØØØ,35ØØ,4ØØØ,45ØØ,11ØØ
 117Ø END
 15ØØ GOSUB7ØØØ:OPEN1:B=Ø:FORJ=ØTO1E8:GET#1,X$:S=ST:X=ASC(X
$)
 151Ø IF(S)THENPRINT:PRINT"STATUS"S:CLOSE1:J=1E9:NEXTJ:GOTO
11ØØ
 152Ø IFX>1THEN16ØØ
 153Ø GET#1,X$:X=ASC(X$)-1:IFX=4THENCLOSE1:PRINT"BIS";J-1:J
```

```
=1E9:NEXTJ:GOTO1100
 1540 IFX<>3THEN1600
 1545 IFJ<=BTHENPRINT"1":PRINT"            1"
 1547 IFJ>BTHENPRINT"BIS";J-1
 1550 INPUT#1,B:PRINT"VON"B;:J=B-1:NEXTJ
 1600 ONTGOTO1700,1800
 1610 NEXTJ
 1700 IFV>=JANDV1<=JTHENPOKEJ,X
 1710 NEXTJ
 1800 IFX<>PEEK(J)THENPRINT:PRINT"?**,FEHLER : LOC."J"B."X"
S."PEEK(J)
 1810 NEXTJ
 3000 IFV1=0THEN1100
 3010 X=INT(V1/256):POKE2,X:POKE1,V1-256*X:PRINT"RESULTAT";
USR(V):GOTO1100
 3500 IFV1>65535THEN1100
 3510 FORJ=V1TOV:X=INT(J/256):POKEIN,J-256*X:POKEIN+1,X:SYS
(IN-3)
 3515 T=(PEEK(C1+5)=255):Z=PEEK(C1):X=PEEK(C1+4)
 3520 PRINTRIGHT$("    "+STR$(J),5);:N=J:GOSUB5:PRINTRIGHT$(
"   $"+H$,6);"   ";
 3530 IFX<25ORTTHENB=1:GOTO3600
 3540 IFX<33THENB=2:GOTO3600
 3550 B=VAL(MID$(B$,Z+1,1))
 3600 FORK=1TOB:N=PEEK(C1+K):GOSUB5:PRINT" "H$;:NEXTK:PRINT
SPC(13-3*B);
 3610 IFTTHEN3800
 3620 FORK=5TO9:PRINTCHR$(PEEK(C1+K));:NEXTK
 3630 IFB=1ANDX<33THEN3800
 3640 IFB>2ORX>32THEN3700
 3650 X=PEEK(C1+2):IFX>127THENX=X-256
 3660 N=X+J+2:GOSUB5:PRINTH$;:GOTO3800
 3700 IFZ=1GOTO3800
 3730 N=PEEK(C1+2)+256*(B-2)*PEEK(C1+3):GOSUB5:PRINTH$;
 3740 IFZ=8THENPRINT")";
 3750 IFZ=4THENPRINT"),Y";
 3760 IFZ=0THENPRINT",X)";
 3770 IFZ=6ORZ=9THENPRINT",Y";
 3780 IFZ=7ORZ=5THENPRINT",X";
 3800 PRINT:J=J+B-1:NEXTJ:GOTO1100
 4000 IFV1>65535THENPRINT"1????1":GOTO1100
 4010 IFV>0ORLEFT$(M$,1)="$"THENPOKEV1,V:GOTO1100
 4020 V2=VAL(RIGHT$(M$,LEN(M$)-3)):T=0:FORI=5TO7:POKEI+C1,A
```

```
SC(MID$(M$,I-4,1))
 4030 NEXTI:POKEC1,V2:SYS3239
 4060 T=PEEK(C1+5):IFT<255THENPOKEV1,T:GOTO1100
 4200 PRINT"??2MODI:0=(*,X) 1=ZE 2=# 3=AB 4=(*),Y"
 4210 PRINT"5=ZE,X 6=AB,Y 7=AB,X 8=(*) 9=ZE,Y 10=AKK"
 4220 GOTO1100
 4500 IFV1=VANDFL=1THENPRINT#1,CHR$(1);CHR$(5):CLOSE1:FL=0:
GOTO1100
 4510 IFV1=VANDFL=0THEN1100
 4520 IFFL=0THENGOSUB7000:FL=1:OPEN1,1,1
 4530 PRINT#1,CHR$(1);CHR$(4);V1:FORJ=V1TOV
 4600 X=PEEK(J):IFX=0ORX=1ORX=10ORX=29THENPRINT#1,CHR$(1);:
X=X+1:Z=Z+1
 4610 PRINT#1,CHR$(X);:Z=Z+1:NEXTJ
 4620 PRINT"? SA000 ,0001":GOTO1100
 7000 PRINT"KASS. 1 ZUM "K$(1+SGN(T-7))" VORBEREITEN;TASTE"
 7010 GETX$:IFX$=""THEN7010
 7020 RETURN
READY.
```

Nachfolgend zeigen wir Ihnen ein kleines Beispielprogramm, welches mit einem CBM und einen Matrixdrucker durchgeführt wurde.
Dabei geht man wie folgt vor:

1. Editor einlesen
2. Programm mit Zeilenzahlen eingeben
3. mit SA Eingabe beenden.
4. Cassette auf Aufnahme schalten
5. Programm abspeichern
6. Cassette zurückspulen
7. CBM ausschalten
8. Assembler einlesen
9. RUN
10. B eingeben
11. Cassette 1
12. 0 drücken
13. 1 drücken
14. Ausgabe auf Drucker (1)
15. mit RS 232 Interfaceplatine und Drucker wird das assemlierte Programm ausgegeben.

Programm mit Fehler:

```
 19 ZEILEN 2 SYMBOLE
033E                1 *=830
033E                2 ! TESTPROGRAMM
033E A000           3 LDY#0
0340 A200           4 LDX#0
0342 B90080         5 MARKE LDA 32768,Y
0345 4900           6 EOR#% 10000000
0347 990080         7 MK1 STA 32768,Y
034A C8             8 INY
034B D0F5           9 BNE MARKE
034D EE4403        10 INC MARKE+2
0350 EE4903        11 INC MK1+2
0353 E8            12 INX
0354 E004          13 CPX #4
0356 D0EA          14 BNE MARKE
0358               15 LDA #32768
```

```
*****FEHLER 2                †
0358 A900       15 LDA #32768
035A 8D4403     16 STA MARKE+2
035D 8D4903     17 STA MK1+2
0360 60         18 RTS
0361            19 .ENDE

MARKE  = 0342         MK1    = 0347
```

Programm ohne Fehler:

```
 19 ZEILEN 2 SYMBOLE
033E             1 *=830
033E             2 ! TESTPROGRAMM
033E A000        3 LDY#0
0340 A200        4 LDX#0
0342 B90080      5 MARKE LDA 32768,Y
0345 4900        6 EOR#% 10000000
0347 990080      7 MK1 STA 32768,Y
034A C8          8 INY
034B D0F5        9 BNE MARKE
034D EE4403     10 INC MARKE+2
0350 EE4903     11 INC MK1+2
0353 E8         12 INX
0354 E004       13 CPX #4
0356 D0EA       14 BNE MARKE
0358 A980       15 LDA #32768>
035A 8D4403     16 STA MARKE+2
035D 8D4903     17 STA MK1+2
0360 60         18 RTS
0361            19 .ENDE

MARKE  = 0342         MK1    = 0347
```

CBM mit Heathdrucker



NOTIZEN

NOTIZEN

Literaturverzeichnis:
National Semiconductors : Preisliste, Datenbücher , Datenblätter.
Signetics: Preisliste und Datenbücher.
Motorola : Master Selection Guide, CMOS Datenbuch, Optoelectronics at work, The Semiconduct. Databook.
RCA : C MOS Datenbuch , Thyristor Datenbuch.
ECC - Interchangeability and Cross Reference Guide
Siemens : Halbleiter Vergleichsliste, Siemens Thyristoren.
Texas Instruments Datenbücher : CC 401, CC 411, CC 416, C MOS - Broschüre.
Texas Instruments Power Databook
Texas Instruments Linear and Interface Databook.
Prospekt der Firma "electronic 2000" München
IC - Datenbuch, Steinbach
Fairchild Datenbücher Linear und Digital
Fairchild Low Power Schottky Datenblatt

Erklärung der verwendeten Abkürzungen für die Hersteller:

Sil.Gen.......... Silicon General
Transitr.......... Transitron
ECC .... Electronic Control Corporation
Gen.El.......... General Electric
Motor........... Motorola
Hew.P. .... Hewlett Packard
NSC, Nat.Se..... National Semiconductors
Fairch........... Fairchild
Siemen.......... Siemens
Signet......... Signetics
Texas I.........Texas Instruments
Telefunk....... Telefunken
Monsan........ Monsanto

Ing.
W. Hofacker
Verlag
8 München75
Postf. 437

Verlagsprogramm

Übersicht lieferbarer Bücher

Bestell Nr.	ISBN	Verfasser	Titel	Preis DM
1	3-921682-01-0	Hofacker	Transistor Berechnungs- und Bauanleitungshandbuch, Band 1	19,80
2	3-921682-02-9	Hofacker	Transistor Berechnungs- und Bauanleitungshandbuch, Band 2	19,80
3	3-921682-03-7	Gebauer	Elektronik im Auto	9,80
4	3-921682-04-5	Lorenz	IC-Handbuch, TTL, CMOS, Linear	19,80
5	3-921682-05-3	Steinbach	IC-Datenbuch, TTL, CMOS, Linear	9,80
6	3-921682-06-1	Steinbach	IC-Schaltungen, TTL, CMOS, Linear	9,80
7	3-921682-33-9	Hofacker	Elektronik Schaltungen	5,–
8	3-921682-08-8	Lorenz	IC-Bauanleitungs-Handbuch	19,80
9	3-921682-09-6	Lorenz	Feldeffekttransistoren	5,–
10	3-921682-34-7	Lorenz	Elektronik und Radio, 4. Auflage	19,80
11	3-921682-11-7	Lorenz	IC-NF Verstärker	9,80
12	3-921682-12-6	Bernstein	Beispiele Integrierter Schaltungen (BIS)	19,80
13	3-921682-13-4	Lorenz	HEH, Hobby Elektronik Handbuch	9,80
14	3-921682-14-2	Lorenz	IC-Vergleichsliste	29,80
15	3-921682-15-0	Lorenz	Optoelektronik Handbuch	19,80
16	3-921682-16-9	Bernstein	CMOS Teil 1, Einführung, Entwurf, Schaltbeispiele	19,80
17	3-921682-17-7	Bernstein	CMOS Teil 2, Entwurf und Schaltbeispiele	19,80
18	3-921682-18-5	Bernstein	CMOS Teil 3, Entwurf und Schaltbeispiele	19,80
19	3-921682-19-3	Lorenz	IC-Experimentier Handbuch	19,80
20	3-921682-20-7	Lorenz	Operationsverstärker	19,80
21	3-921682-21-5	Lorenz	Digitaltechnik Grundkurs	19,80
22	3-921682-22-3	Bernstein	Mikroprozessoren, Eigenschaften und Aufbau 2. Aufl.	19,80
23	3-921682-23-1	Lorenz	Elektronik Grundkurs, Kurzlehrgang Elektronik	9,80
24	3-921682-35-5	Hans Peter Blomeyer-Bartenstein	Mikrocomputer Technik	29,80
25	3-921682-25-8	C. Lorenz	Hobby Computer Handbuch	29,80
26	3-921682-26-8	H. Bernstein	Mikroprozessor Teil 2	19,80
27	3-921682-27-4	C. Lorenz	Mikrocomputer Software Handbuch	29,80
28	3-921682-28-2	C. Lorenz	Lexikon + Wörterbuch für Elektronik und Mikroprozessortechnik LEM	29,80
29	3-921682-29-0	C. Lorenz	Mikrocomputer Datenbuch	49,80
30	3-921682-30-4	C. Lorenz	Aktivtraining-Mikrocomputer	49,80
31	3-921682-31-2	C. Lorenz	57 Programme in BASIC	39,–
32	3-921682-32-0	C. Lorenz	ATARI BASIC Handbuch	29,80
33	3-921682-36-6	Dr. Hatzenbichler	Microcomputer Programmierbeispiele	19,80
34	3-921682-50-7	H. Hermann	TINY-BASIC Handbuch	19,80
35	3-921682-35-5		Der freundliche Computer	29,80
41	–	–	Experimentierplatine für 14, 16, 24, 28 und 40 polige DIL IC's	79,–
1051	–	–	TTL-Experimentierbuch	5,–
1061	–	–	CMOS-Experimentierbuch	5,–
108	3-921682-42-8	C. Lorenz	SC/MP Mikrocomputer-Handbuch	29,80
109	3-921682-43-6	P. Heuer	6502 Microcomputer Programmierung	29,80
110	3-921682-49-5	C. Lorenz	Programmierhandbuch für PET	29,80
111	3-921682-45-2	M. Stübs	Programmieren mit TRS-80	29,80
113	3-921682-48-7	C. Lorenz	BASIC Programmierhandbuch	19,80
114	3-921682-60-6	L. Oswald	Der Microcomputer im Kleinbetrieb	39,80
118	3-921682-61-4	C. Lorenz u. R. Lullus	Programmieren in Maschinensprache mit dem 6502	98,–
119	3-921682-62-2	C. Lorenz	Programmieren in Maschinensprache (Z80)	49,–
120	3-921682-64-9	M. Stübs	Anwenderprogramme für TRS-80	29,80
150	3-921682-52-5		Care and Feeding of the Commodore PET (engl.)	19,80
151	3-921682-51-7		8K Microsoft BASIC Reference Manual (engl.)	19,80
152	3-921682-67-3	S. Roberts	Expansion Handbook for 6502 and 6800 (engl.)	19,80
153	3-921682-54-1		Microcomputer Application Notes (engl.)	29,80
154	3-921682-53-3		Complex Sound Generation using the SN76477 (engl.)	19,80
155	3-921682-56-8		The First Book of 80-US(TRS-80) (engl.)	19,80
156	3-921682-68-1	S. Roberts	Small Business Programs (engl.)	29,80

Ing. W. Hofacker GmbH Verlag 8 München 75